
Digitized by the Internet Archive
in 2013

http://archive.org/details/levelrestrictedn849hukc

ort No. UIUCDCS-R-77-849 UILU-ENG 77 1703

LEVEL-RESTRICTED NOR NETWORK TRANSDUCTION PROCEDURES

by

K. C. Hu

uary, 1977

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-77-849

LEVEL-RESTRICTED NOR NETWORK TRANSDUCTION PROCEDURES

by

K. C. Hu

January, 1977

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

This work was supported in part by the Department of Computer Science and by the National Science Foundation under Grant No. DCR73-03421 A01.

ACKNOWLEDGMENT

The author is grateful to Professor S. Muroga for his guidance and discussions during the work presented in this report and also for his careful reading and valuable suggestions for the improvement of the original manuscript.

The author is also greatly indebted to H. C. Lai and J. N. Culliney for their invaluable discussions.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

The synthesis of near-optimal NOR (or NAND) networks is one of important problems in logic design. Recently, NOR network transduction procedures were developed to solve this problem [1,2,8,9,10,11,12,13,14,15].

"Transduction" means transformation and reduction. Most of the transduction procedures use the concept of compatible set of permissible functions (CSPF) [9]. For any given non-optimal network, some transduction procedures can remove redundant gates and/or connections and some transduction procedures can also reconfigure the network by connecting external variables or internal connections to some gates so that a near-optimal network can be obtained.

Existing transduction procedures can be grouped into three classes, according to their characteristics:

(a) Pruning procedures: These types of transduction procedures try to remove redundant connections (external variable connections or internal connections) only. But because of the removal of redundant connections, some gates may become redundant. Three programs are implemented to realize these procedures. They are NETTRA-P1, NETTRA-P2 and NETTRA-PG1 (with parameter set to 0*) [1,14].

---

*NETTRA-PG1 has a parameter I. It can be either 0 or 1. When I is 0, a pruning procedure is implied. When I is 1, a substitution procedure is implied.

(b) General procedures: These types of transduction procedures can either add or remove connections in a network, based on the concept of compatible set of permissible functions. Four programs realize these procedures. They are NETTRA-G1, NETTRA-G2, NETTRA-G3, NETTRA-G4 and NETTRA-PG1 (with parameter set to 1) [1,2,8,11,13,14].

(c) Error-compensation procedures: These procedures remove a gate at a time and try to compensate for the changes in the output function(s) by reconfiguring the network. Programs NETTRA-E1 and NETTRA-E2 realize these procedures [7,10,12].

All of the above nine programs, except NETTRA-G4, can treat fan-in/fan-out restricted problems*. In order to consider the number of levels in a network also as a restriction, procedures based on gate substitution (in classes a and b), gate merging (in class b), connectable and disconnectable functions (in class b) and error-compensation (in class c) are modified. Then a computer program is designed to organize these procedures together to try to solve problems under both fan-in/fan-out and level restrictions. In this paper, the modification of the transduction procedures for level restriction is explained, along with the outline of the computer program.

The contents of this report are as follows. In Chapter 2, the basic ideas which are used in designing level-restricted networks are given. Network transduction procedures usually start from an initial network, i.e., a given network which we want to simplify (it can be derived by any conventional design procedure and is not necessarily optimal). Initial networks used for

---

*Programs NETTRA-G1, NETTRA-G2, NETTRA-G3, NETTRA-E1, NETTRA-E2 and NETTRA-PG1 were modified to treat fan-in/fanout restrictions. The remaining programs realize pruning procedures, so they will not cause any fan-in/fan-out problems if the given network is already fan-in/fan-out restricted.

problems with both level restriction and fan-in/fan-out restrictions are different from those used for problems with only fan-in/fan-out restrictions. Section 3.1 presents the basic ideas of finding level-restricted initial networks. Section 3.2 explains some practical consideration in programming the idea mentioned in section 3.1. Section 3.3 gives the results of some experiments which show the effectiveness of the initial network subroutine. Chapter 4 provides the explanation of the modification of the transduction procedures for level restriction. In each section, the basic principles of the transduction procedures are reviewed first and then the modifications are explained.

Chapter 5 shows the results of experiments of level-restricted transduction programs. Section 5.1 outlines the organization of the main control subroutine. Section 5.2 gives the results of many sample examples under different combinations of fan-in/fan-out and level restrictions. In section 5.3, the level-restricted transduction program is compared with the logic design program based on the branch-and-bound method [18,19].

CHAPTER 2
BASIC IDEAS

For any given switching function, there always exist optimal NOR networks if there are only fan-in/fan-out restrictions.* This is also true if the number of levels is the only constraint.† But the fan-in/fan-out restrictions and the level restriction tend to contradict each other. Since solving a fan-in/fan-out problem will usually increase the number of levels and reducing the number of levels will usually increase fan-outs of gates and external variables or fan-ins of gates. So if both fan-in/fan-out restrictions and level restriction are simultaneously imposed, it is not guaranteed that a feasible network (i.e., a network which realizes the given functions under given restrictions) can be obtained. An example is shown in Figure 2.1. The functions in this example are the sum and the carry of the one-bit full adder. If the maximum fan-in of each gate (FI) and the maximum fan-out of each external variable (FOX) and the maximum fan-out of each gate (FO) are 3, and if the maximum number of levels (LEV) is 100 (i.e., we choose a large number 100 such that no restriction is imposed on the number of levels), then we can get an optimal network in Figure 2.1(a) which has

---

*This is always true if: (i) the maximum fan-in of each gate is greater than or equal to 2, (ii) the maximum fan-out of each external variable is greater than or equal to 2, and (iii) the maximum fan-out of each gate is greater than or equal to 1. We are not going to prove this here.

†This is always true if the maximum number of levels is greater than or equal to 2 (if both complemented and uncomplemented external variables are permitted) or 3 (if only uncomplemented external variables are permitted).

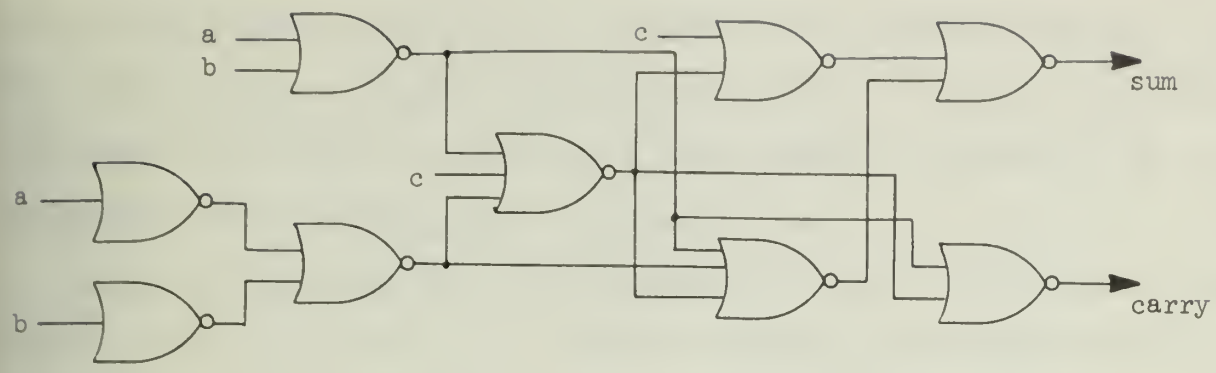


Fig. 2.1(a) Optimal one-bit full adder under restrictions LEV=100 and FI=FOX=FO=3.

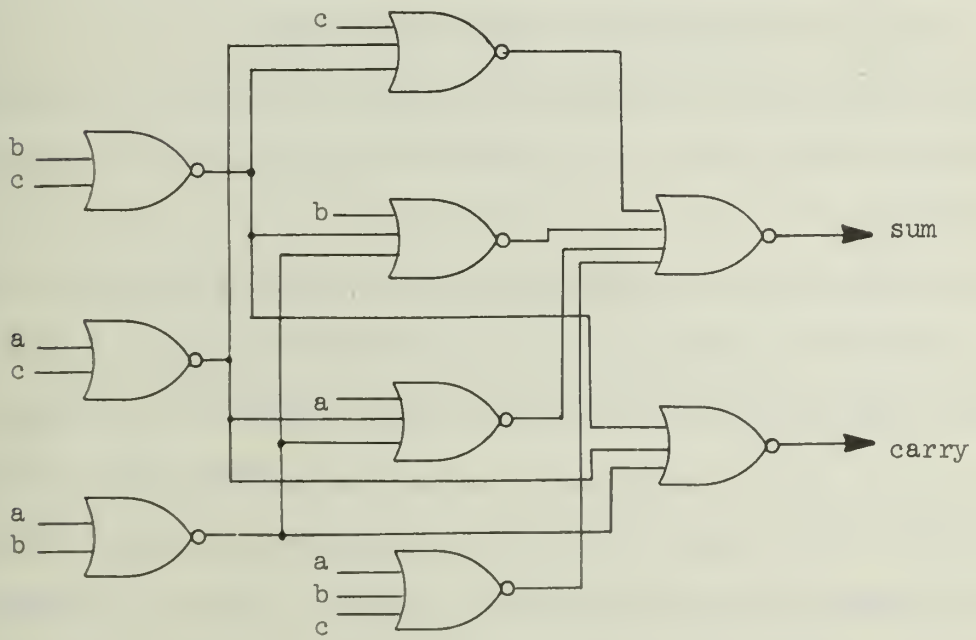


Fig. 2.1(b) Optimal one-bit full adder under restrictions LEV=3 and FI=FOX=FO=100.

9 gates and 18 connections. If FI = FOX = FO = 100 (i.e., no fan-in/fan-out restrictions) and LEV = 3, we can get an optimal network in Figure 2.1(b) which has 9 gates and 25 connections. But if we set FI = FOX = FO = 3 and LEV = 3, we cannot get any feasible network. The optimality of the networks in Figure 2.1(a) and (b) and the infeasibility of the above case are proved by the branch-and-bound algorithm.

Network transduction procedures can also produce near-optimal networks if only one of fan-in/fan-out restrictions and level restriction* is imposed. The following procedures are usually followed to try to find near-optimal networks:

(a) Find a non-optimal initial network for the given switching function.

(b) Apply transduction procedures to try to simplify the initial network as much as possible (no fan-in/fan-out restrictions are considered).

(c) Apply transformation procedures to transform the network obtained in step (b) into a fan-in/fan-out restricted network.

(d) Apply fan-in/fan-out restricted transduction procedures to simplify the network obtained in step (c) without violating the fan-in/fan-out restrictions.

For step (a), five different methods are being considered to produce initial networks: the branch-and-bound algorithm [18,19,20,21], the three-level networks based on the map factoring method [16], the Gimpel algorithm [4,5], the universal network method [16] and the Tison method [3]. Although the initial networks usually have redundant gates and/or connections, they can be produced within a very short time. For step (b), any transduction

---

*The two-level or three-level networks (depending on whether both complemented and uncomplemented external variables are permitted) can be obtained by Tison's Method [3] or Gimpel's algorithm [4,5].

procedure can be used. For step (c), the network transformation method designed by Jeffry G. Legge can be employed [15]. For step (d), the pruning procedures and those procedures which have been modified for fan-in/fan-out restrictions can be applied.

When the level restriction is additionally imposed, the above procedures must be modified for the following reasons. First, the initial networks obtained in step (a) may not be level-restricted and it is not known whether the number of levels can be reduced by the transduction procedures. Second, the transformation procedure in step (c) will usually increase the number of levels significantly. Third, the transduction procedures may also increase the number of levels.

The basic ideas used in obtaining near-optimal networks under both fan-in/fan-out restrictions and level restriction are as follows:

(1) Find an initial network which satisfies the level restriction.

(2) Apply the modified transduction procedures (modification will be discussed later) to simplify the initial network without violating the level restriction. If the initial network is already fan-in/fan-out restricted, then it could be simplified into a feasible near-optimal network. The transduction procedures may solve some fan-in/fan-out problems, if the initial network is not fan-in/fan-out restricted. If all fan-in/fan-out problems can be solved, then a feasible near-optimal network can be obtained.

Obviously, this approach does not guarantee that feasible networks (i.e., networks which realize the given functions under given restrictions) can be found even if there do exist optimal networks for the given problem. But the statistics (Chapter 5) show that feasible networks can be obtained

in most cases and the results derived by this approach are reasonably good, especially in the case that the maximum number of levels is not very small.

The above approach needs a method which can produce level-restricted initial networks. It also needs transduction procedures which can treat level-restricted problems. The implementation of initial networks will be explained in Chapter 3. The modification of transduction procedures for level restriction will be given in Chapter 4.

CHAPTER 3
INITIAL NETWORKS FOR LEVEL-RESTRICTED TRANSDUCTION PROCEDURES

It has been well-known that there always exists a two-level (if both complemented and uncomplemented external variables are available as network inputs) or a three-level (if only uncomplemented external variables are available) NOR network* realizing a given switching function. This initial network is always level-restricted. But since it may have fan-in/fan-out problems, we may need more levels. The level-restricted initial networks for the network transduction procedures are obtained by increasing the number of levels of a two-level or a three-level network to the maximum limit while solving the fan-in/fan-out problems as many as possible. Since there exists a minimum product (i.e., minimum product of alterms) for any switching function, we use the 2-level or 3-level network based on the minimum product to start with.

3.1 General Description

Assume that a minimum product for the given switching function f consists of ℓ alterms, and the i-th alterm ($1 \leq i \leq \ell$) consists of n_i literals. A two-level NOR network can be obtained (for convenience, assume both complemented and uncomplemented external variables are permitted) in Figure 3.1.

---

*For example, the network based on any conjunctive form (i.e., a two-level network of AND and OR gates is obtained from this and then converted into a NOR gate network) or the network obtained by Gimpel's algorithm [4] is 2- or 3-level.

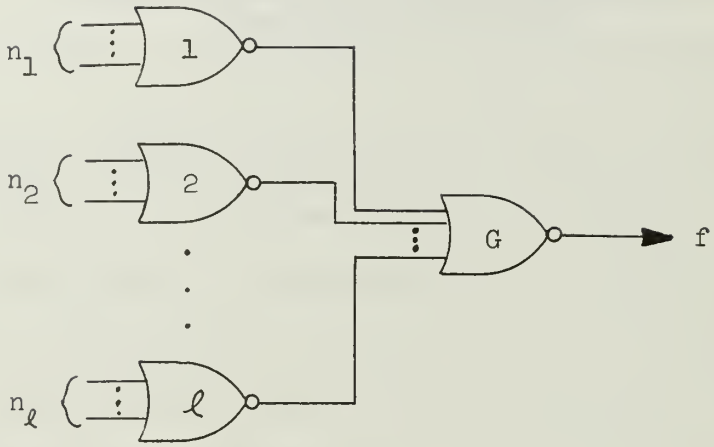


Fig. 3.1 Two-level network based on the minimum product.

Now, let us consider the fan-in/fan-out restrictions. Let the maximum fan-in of a gate be FI, the maximum fan-out of an external variable be FOX and the maximum fan-out of a gate be FO. In Figure 3.1, if $\ell \leq$ FI, then there is no fan-in problem at the first level gate G, and we can go to check the fan-in problems of the higher level gates. If $\ell >$ FI, then we can solve this fan-in problem using the following approach:

(1) Partition the input functions of gate G into at most FI groups.

(2) Realize the disjunction of the functions of each group by a two-level subnetwork.

(1) solves the fan-in problem of gate G. (2) increases the number of levels by one, and may generate some fan-in/fan-out problems in those two-level subnetworks. But the fan-in problems can be solved by applying procedures (1) and (2) repeatedly. It is easy to see that networks derived by this approach will have a tree structure. Hence, there is no fan-out problems for gates. There may be fan-out problems for external variables, but they can be solved by adding extra inverters.

Following the above procedures, we can always get a level-restricted network: since each time these procedures are applied, the number of levels of the network is increased at most by 1.* We can also follow these procedures repeatedly until a fan-in/fan-out restricted network is obtained.

In procedure (1), how to divide the ℓ input functions into FI groups is an important problem. An improper formation of groups may generate more fan-in/fan-out problems in higher level gates. Consider the network shown in Figure 3.2(a). Suppose the disjunction of functions $f_1, f_2, \ldots,$ and f_k are to be realized by a two-level single-output subnetwork, where f_i is the output function of gate i which is fed by n_i inputs (external variables). In the worst case, all inputs (external variables) of $f_1, f_2, \ldots, f_k$ are different. Assume they are $x_1, x_2, \ldots, x_{n_1+n_2+\ldots+n_k}$ (for simplicity, assume all uncomplemented). Then

$$\bigvee_{i=1}^{k} f_i = \bar{x}_1\bar{x}_2\ldots\bar{x}_{n_1} \vee \bar{x}_{n_1+1} \ldots \bar{x}_{n_1+n_2} \vee \ldots$$

$$\vee \bar{x}_{n_1+n_2+\ldots+n_{k-1}+1}\ldots\bar{x}_{n_1+n_2+\ldots+n_k}$$

---

*In the case where only uncomplemented external variables are available, the number of levels may be increased by two after applying (2) if there exists no inverter having an external variable as its input in the original network.

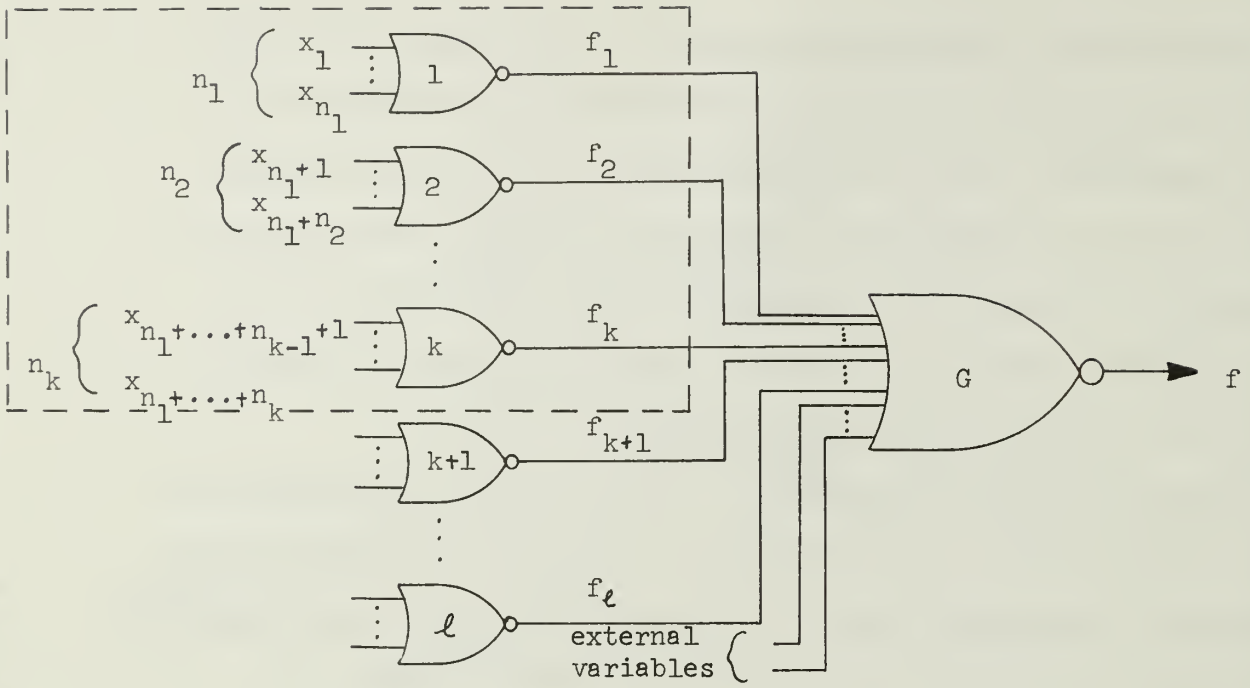


Fig. 3.2(a) $\bigvee_{i=1}^{k} f_i$ is to be realized by a two-level single-output subnetwork.

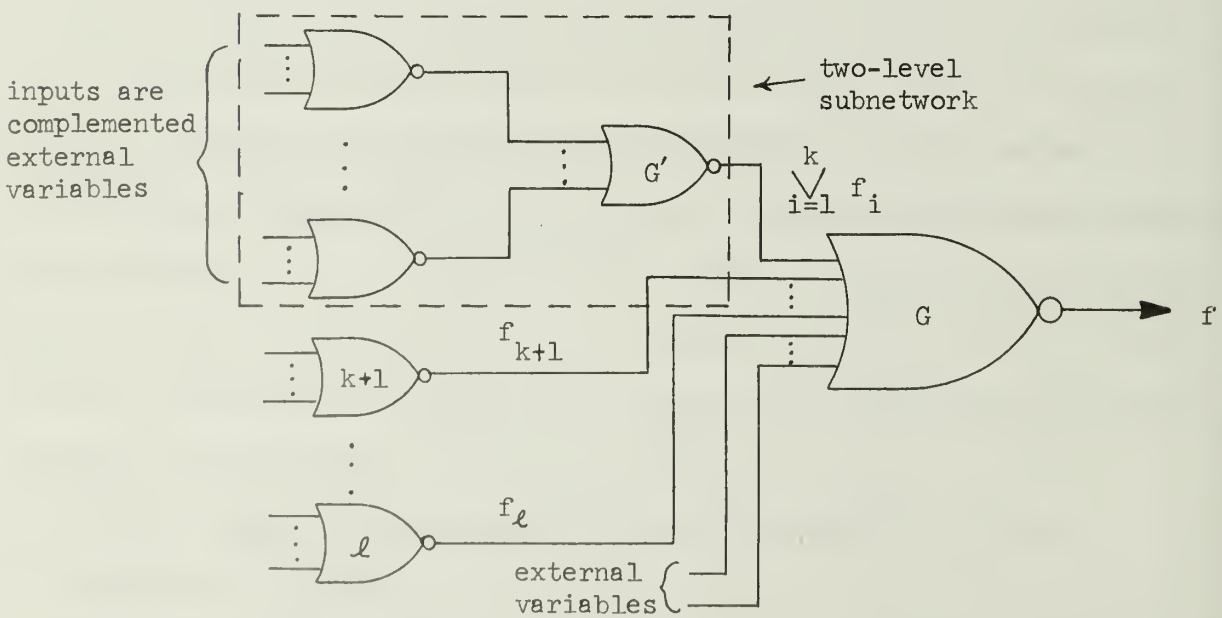


Fig. 3.2(b) The result after applying procedures (1) and (2).
There are $\prod_{i=1}^{k} n_i+1$ gates in the two-level subnetwork.

or

$$\overline{\bigvee_{i=1}^{k} f_i} = (x_1 \vee x_2 \vee \ldots \vee x_{n_1}) \cdot (x_{n_1+1} \vee \ldots \vee x_{n_1+n_2}) \ldots$$

$$(x_{n_1+n_2+\ldots+n_{k-1}+1} \vee \ldots \vee x_{n_1+n_2+\ldots+n_k}) \qquad (3.1.1)$$

If we multiply out equation (3.1.1), we can get a disjunctive form which has $\prod_{i=1}^{k} n_i$ terms and each term is a product of k literals. Taking the complement of this disjunctive form, we can get a conjunctive form for $\bigvee_{i=1}^{k} f_i$. This conjunctive form has $\prod_{i=1}^{k} n_i$ alterms and each alterm is a disjunction of k literals. Apparently, a two-level NOR subnetwork can be obtained based on this conjunctive form.

The total number of NOR gates in this subnetwork is $\prod_{i-1}^{k} n_i + 1$: $\prod_{i=1}^{k} n_i$ gates in the higher level and another one in the output level, see Figure 3.2(b). An example is given below.

Assume $f_1 = \bar{x}_1\bar{x}_2$, $f_2 = \bar{x}_3\bar{x}_4$, i.e., $n_1 = 2$ and $n_2 = 2$

$$\overline{f_1 \vee f_2} = (x_1 \vee x_2)(x_3 \vee x_4)$$

$$= x_1x_3 \vee x_1x_4 \vee x_2x_3 \vee x_2x_4$$

$$= \overline{(\bar{x}_1 \vee \bar{x}_3)(\bar{x}_1 \vee \bar{x}_4)(\bar{x}_2 \vee \bar{x}_3)(\bar{x}_2 \vee \bar{x}_4)}$$

So a two-level subnetwork can be formed as in Figure 3.3(b). Figure 3.3(a) shows the original network for $f_1 \vee f_2$.

Of course, the above analysis is the worst case. Usually some x_i and $\bar{x}_i$ both appear in equation (3.1.1) so that many terms become identically zero after multiplying out equation (3.1.1)--this means that the total number of

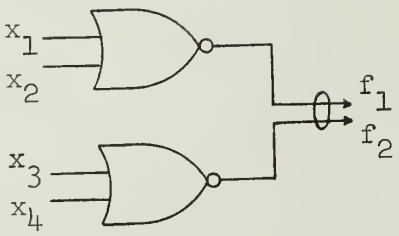


Fig. 3.3(a)

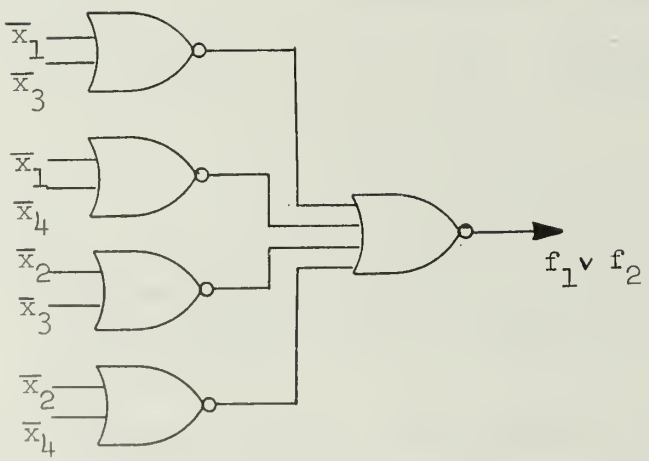


Fig. 3.3(b)

the second-level gates or the total number of fan-in of gate G' is less than $\prod_{i=1}^{k} n_i$. Sometimes x_i (or $\bar{x}_i$) appears in more than one alterm in equation (3.1.1) so that many terms may contain fewer literals than k and also some terms subsume other terms and hence can be eliminated after multiplying out equation (3.1.1). In general, if more pairs of x_i and $\bar{x}_i$ appear in different alterms and x_i (or $\bar{x}_i$) appears in more alterms, then fewer gates will have fan-in problems and fewer external variables will have fan-out problems in the resulting two-level subnetwork. For example, consider the two-level network in Figure 3.4(a), where $f_1 = x_1x_2\bar{x}_3$, $f_2 = \bar{x}_1\bar{x}_2\bar{x}_3$, $f_3 = x_1x_4x_5$ and FI = FOX = 2. Since the output gate has 3 inputs, let us first solve this fan-in problem. It is obvious that if we realize the disjunction of any two of the three functions f_1, f_2 and f_3 by a two-level subnetwork, we can reduce the fan-in of the output gate by 1. Figure 3.4(b) through Figure 3.4(d) show the results of three possible ways of partitioning f_1, f_2 and f_3 into two groups. In Figure 3.4(b), $f_1 \vee f_2$ is realized. $f_1 \vee f_2$ has two pairs of complemented and uncomplemented external variables and $\bar{x}_3$ appears twice. In Figure 3.4(c), $f_1 \vee f_3$ is realized. $f_1 \vee f_3$ has no complemented and uncomplemented pairs but x_1 appears twice. In Figure 3.4(d), $f_2 \vee f_3$ is realized. $f_2 \vee f_3$ has one complemented and uncomplemented pair but no literal appears two or more times. Obviously, the subnetwork obtained in Figure 3.4(b) is the best among three. It only has three gates and the output gate of the subnetwork only has three inputs. Since this network still has fan-in problems, we apply the similar procedures to solve these problems. Then we get the fan-in/fan-out restricted network shown in Figure 3.5 which has 4 levels.

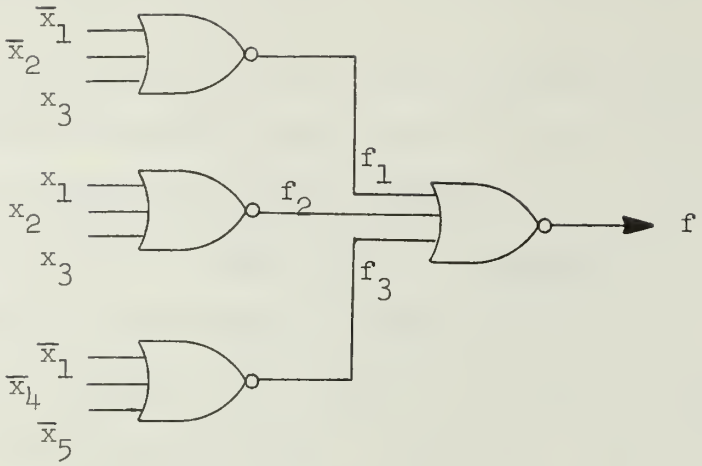


Fig. 3.4(a) The original two-level network.

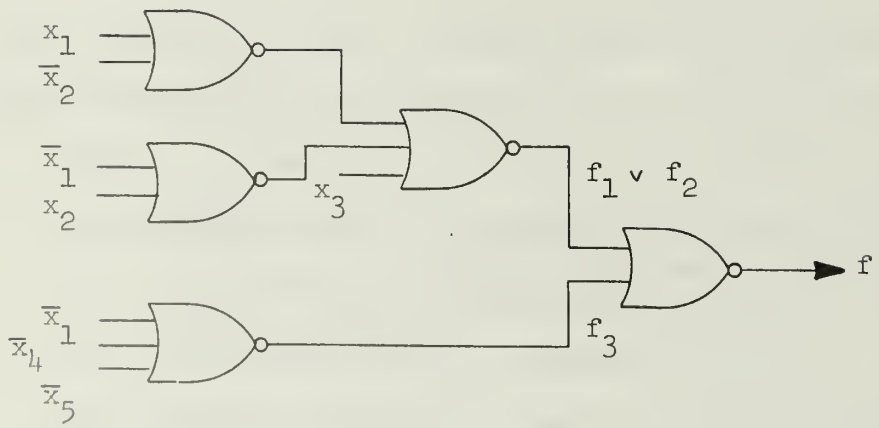


Fig. 3.4(b) The network after realizing $f_1 \vee f_2$.

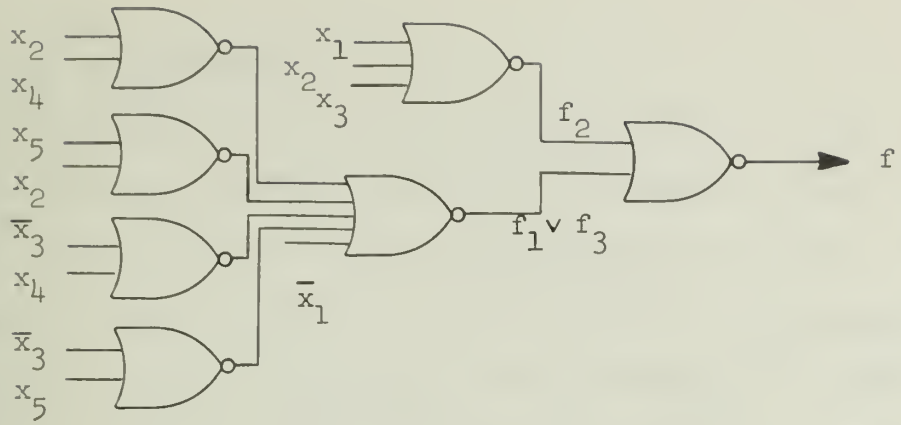


Fig. 3.4(c) The network after realizing $f_1 \vee f_3$.

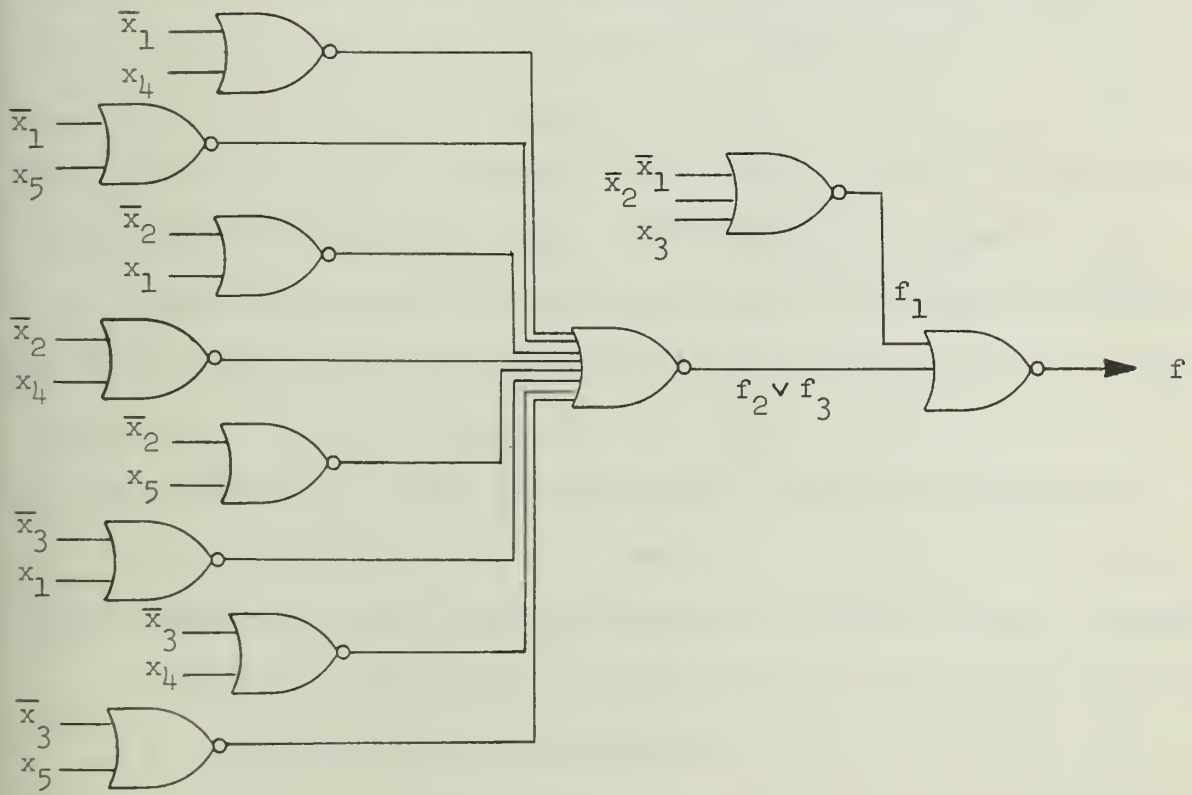


Fig. 3.4(d) The network after realizing $f_2 \vee f_3$.

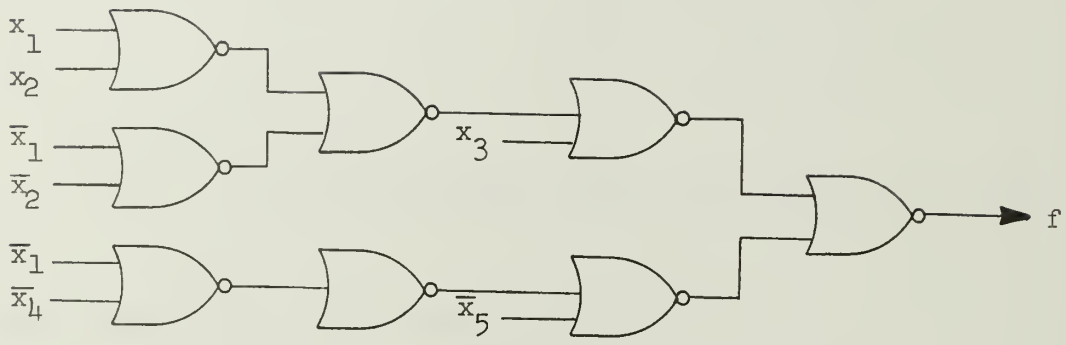


Fig. 3.5 The fan-in/fan-out restricted network for f.

It is observed that in general if there are more common literals and if there are more complemented and uncomplemented literals appearing in equation (3.1.1), then the resultant two-level subnetwork will have a fewer numbe of gates and also fewer gates will have fan-in problems.

Two criteria are used in partitioning the input functions of a gate into groups to solve the fan-in problems:

Criterion 1: Select the group which has the largest number of common literals.

Criterion 2: Select the group which has the largest number of pairs of complemented and uncomplemented external variables.

3.2 Practical Consideration

In programming the procedures for partitioning the input functions of a gate, many practical problems come up. Let the number of input functions to be partitioned be denoted by NUM and the maximum fan-in by FI. Then at most FI groups should be formed out of NUM functions. How to partition NUM input functions into FI groups is not a trivial problem. For example, if NUM = 10 and FI = 2, then there are five possible ways to do the partition: (1,9), (2,8), (3,7), (4,6) and (5,5), where (i,10-i) means that i elements are in one group and the remaining 10-i elements are in the other group. If NUM is larger, the number of groups is larger. If FI is larger, the situation is more complicated. For example, if NUM is still 10 and FI is 3, then there are another eight possible ways to do the partition: (1,1,8), (1,2,7), (1,3,6), (1,4,5), (2,2,6), (2,3,5), (2,4,4) and (3,3,4), besides the five for FI = 2, where (i,j,10-i-j) means that i elements are in the first group, j elements are in the second group and the remaining 10-i-j elements are in the third group. It is impossible to decide which way is the best unless all possible ways have been tried. The approach currently programmed to implement procedure (1) in section 3.1 is explained by the flowchart shown in Figure 3.6. The blocks in this flowchart work as follows:

Block 1: Set NO = 0, where NO is used to count the number of groups that have been formed.

Block 2: Check whether only two functions are required to be grouped together, i.e., whether NUM-FI is 1 or not. If this is true, then go to block 3; otherwise go to block 4.

Block 3: Find a couple of functions which have the largest number of common literals. If there are two or more such couples, choose the couple which has the largest number of pairs of x_i and $\bar{x}_i$. If there are

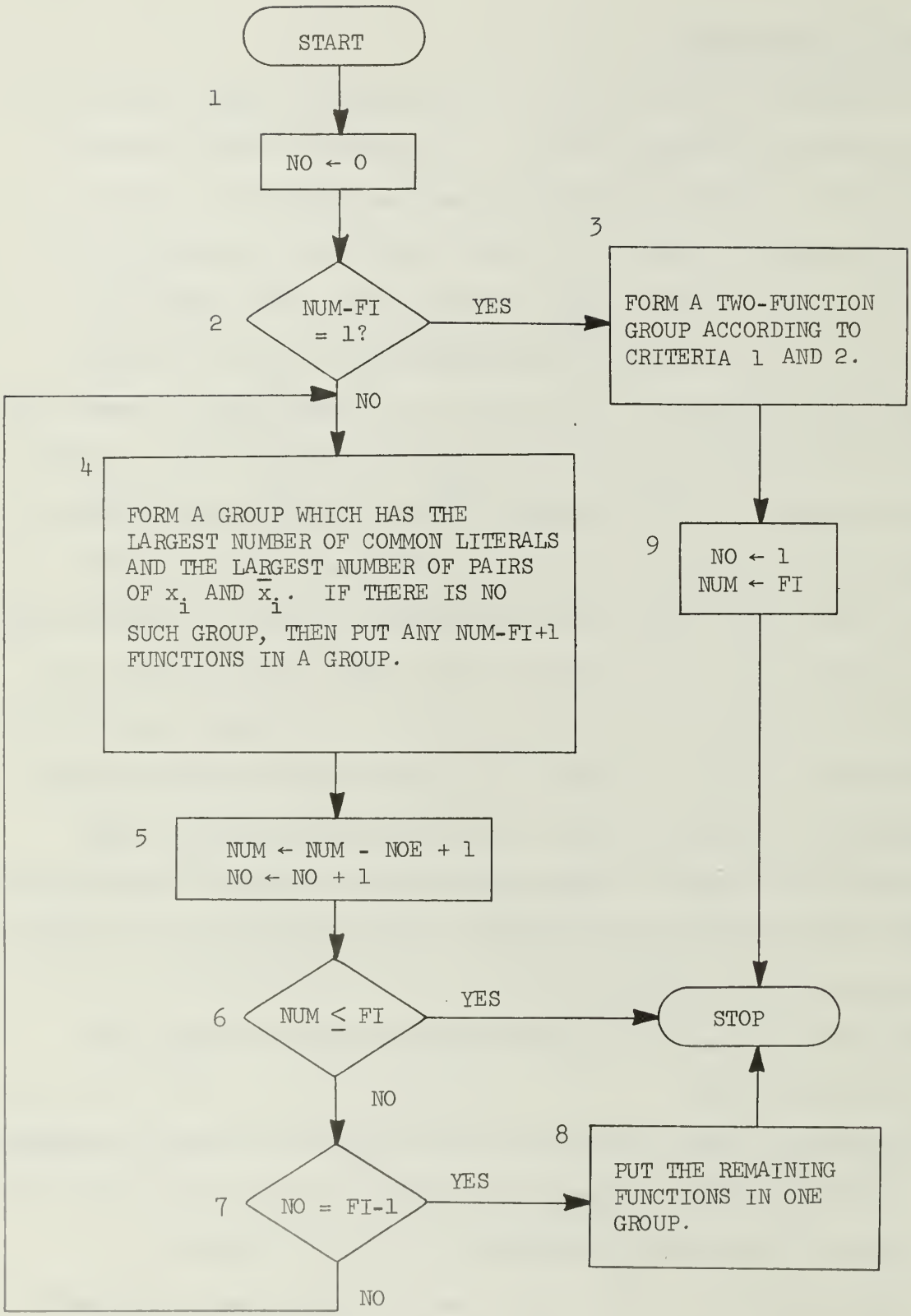


Figure 3.6. Flowchart for partitioning the inputs of a gate into at most FI groups.

two or more such couples, choose any one. If there are no two functions which have common literals and pairs of x_i and $\bar{x}_i$, then choose any two. Go to block 9.

Block 4: Try to form a group in which there is only one common literal and the number of functions in this group is the largest. If there exist two or more such groups, choose the one which has the largest number of pairs of x_i and $\bar{x}_i$. If there are two or more such groups, choose any one. If there does not exist any group which has common literals and pairs of x_i and $\bar{x}_i$, then put any NUM-FI+1 functions in a group.

Block 5: NUM, the number of current inputs of the gate under consideration, is updated as NUM-NOE+1, where NOE is the number of functions used in forming the group in block 4. NO, the number of groups formed, is increased by 1.

Block 6: Check whether NUM $\leq$ FI or not. If NUM is already less than or equal to FI, then stop; otherwise go to block 7.

Block 7: Check whether NO is equal to FI-1. If this is true, go to block 8. If this is not true, i.e., NO < FI-1, then go back to block 4.

Block 8: Put the remaining functions, which have not been used in forming groups, in one group and stop.

Block 9: Set NO to be 1 and NUM to be FI and stop.

In implementing procedure (2) in section 3.1, equation (3.1.1) has to be multiplied out. Usually some redundant terms occur in the multiplied-out form, and each redundant term corresponds to a redundant gate in the two-level subnetwork. For example, if the equation $(x_1 \vee x_2 \vee \bar{x}_3)(x_1 \vee \bar{x}_4 \vee x_3)$ is multiplied out, it will have 9 terms: $x_1 \vee x_1\bar{x}_4 \vee x_1x_3 \vee x_2x_1 \vee x_2\bar{x}_4 \vee x_2x_3 \vee \bar{x}_3x_1 \vee \bar{x}_3\bar{x}_4 \vee \bar{x}_3x_3$. Obviously, $\bar{x}_3x_3$ is identically zero and can be removed. Single-literal term x_1 is subsumed by terms $x_1\bar{x}_4$, x_1x_3, x_2x_1 and $\bar{x}_3x_1$, so the latter four terms can be eliminated. Besides, term $x_2\bar{x}_4$ is the consensus of terms $\bar{x}_3\bar{x}_4$ and x_2x_3, and therefore can be eliminated too. The simplified form contains only three terms: x_1, x_2x_3 and $\bar{x}_3\bar{x}_4$. This means that only three gates are needed in forming the two-level subnetwork: two gates for terms x_2x_3 and $\bar{x}_3\bar{x}_4$ and one gate for the output of the subnetwork (no gate is needed for the single-literal term).

Redundancy check is included in implementing procedure (2) to eliminate terms which are identically zero, terms which subsume other terms and terms which are the consensus of other terms.

In multiple-output network design problems, a tree-structured network will be constructed for each output function if procedures (1) and (2) in section 3.1 are applied. Since usually some gates in different trees realize the same function, they can be replaced by a single gate which feeds to different trees. The fan-in/fan-out restricted and level-restricted transduction procedure based on gate merging is applied to merge or substitute for gates so that a compact initial network can be obtained. Each time an output function is realized by a tree network, the transduction

procedure is employed to merge or substitute for gates in this tree network and the networks implemented for output functions which have been realized already.*

3.3 Effectiveness of the Initial Network Subroutine TISLEV

A computer subroutine TISLEV is designed to obtain initial networks under level restriction. For any given switching function, the Tison's method [3] is applied first to get a minimal product. Starting from this minimal product, subroutine TISLEV is then called to solve the fan-in/fan-out problems in the two-level NOR network designed based on this minimal product, without violating the level restriction.

Ten 4-variable single-output functions, ten 5-variable single-output functions and four multiple-output functions are used to test the effectiveness of this initial network subroutine.

Table 3.3.1 through Table 3.3.9 give the results. In each of these tables, the maximum fan-in/fan-out and the availability of complemented and uncomplemented external variables are shown in the first row. The hexadecimal representation of the truth of functions are given in the first column. The cost of a network is represented by $1000 \times R + C$ where R is the number of gates and C is the number of connections. In Table 3.3.1 and Table 3.3.2, the networks produced by TISLEV under different level restrictions

*In the current version of transduction programs, a maximum 60 gates network is allowed. If we do not apply transduction procedure to simplify the tree network, the total number of gates may be greater than 60.

Table 3.3.1 10 Four-variable Functions

RESTRICTION / COST & TIME / FUNCTION (HEX)	FI = FO = FOX = FOO = 3; uncomplemented inputs only			
	BY TISLEV			BY JEFF
	LEVLIM = 4†	LEVLIM = 5	LEVLIM = 100	(LEVLIM = 100)
1. 4AF1	10020	10020	10020(4)	11022(5)
2. FBFE	8014	8014	8014(4)	8014(4)
3. ABCE	11022*	11021	11021(5)	13024(5)
4. 2D8B	15031*	16030	16030(6)	18035(7)
5. 9DA5	9018	9018	9018(4)	11022(5)
6. 5F12	7013	7013	7013(3)	7013(3)
7. F1F4	7014	7014	7014(3)	7014(3)
8. 6830	12023	12023	12023(4)	12023(5)
9. 9048	14027*	15028	15028(5)	15028(5)
10. EA9B	10020	10020	10020(5)	13025(5)
COMPUTATION TIME IN CENTISECONDS				
1. 4AF1	22	25	30	30
2. FBFE	17	12	14	18
3. ABCE	24	20	22	27
4. 2D8B	30	29	35	29
5. 9DA5	25	17	15	18
6. 5F12	19	10	10	18
7. F1F4	17	10	12	11
8. 6830	30	20	20	27
9. 9048	27	24	24	29
10. EA9B	29	22	20	20

*Not fan-in/fan-out restricted.

†LEVLIM is the maximum number of levels.

Table 3.3.2 10 Five-variable Functions

RESTRICTION / COST & TIME / FUNCTION (HEX)	FI = FO = FOX = FOO = 4 complemented and uncomplemented inputs			
	BY TISLEV			BY JEFF (LEVLIM = 100)
	LEVLIM = 3†	LEVLIM = 4	LEVLIM = 100	
1. 4FA295F6	14041*	14039	14039(4)	15046(6)
2. A6CDDF18	17048*	16044	16044(4)	15046(6)
3. FF68A1F3	18048*	19046	19046(5)	12039(6)
4. 1EE65240	16041*	16040	16040(4)	10032(4)
5. 9E63BE75	13038*	18043	18043(4)	11034(4)
6. 0A888103	11027	11027	11027(3)	10028(4)
7. 49F363CD	14041*	14039	14039(4)	15046(6)
8. 8B5809F0	11033	11033	11033(3)	13038(4)
9. BFD8C6DA	20053*	19049	19049(5)	17049(6)
10. C6E7103E	14037*	14036	14036(4)	10035(4)
COMPUTATION TIME IN CENTISECONDS				
1. 4FA295F6	54	65	64	63
2. A6CDDF18	55	60	62	71
3. FF68A1F3	50	53	55	44
4. 1EE65240	59	59	62	105
5. 9E63BE75	44	45	49	35
6. 0A888103	47	45	47	102
7. 49F363CD	67	75	77	97
8. 8B5809F0	59	62	65	119
9. BFD8C6DA	59	55	64	43
10. C6E7103E	57	55	57	77

*Not fan-in/fan-out restricted. †LEVLIM is the maximum number of levels.

Table 3.3.3 One-bit Full Adder

RESTRICTIONS / COST & TIME / FUNCTION (HEX)	FI = FO = FOX = FOO = 3 uncomplemented inputs only	
	TISLEV LEVLIM $\geq$ 4	JEFF (LEVLIM = 100)
1. SUM 69	15029(4)	17033(6)
2. CARRY 17	52 CS†	27 CS

†CS = centiseconds

Table 3.3.4 One-bit Full Adder

RESTRICTIONS / COST & TIME / FUNCTION (HEX)	FI = FO = FOX = FOO = 3 complemented and uncomplemented inputs	
	TISLEV LELLIM $\geq$ 3	JEFF (LEVLIM = 100)
1. SUM 69	8020(3)	15030(6)
2. CARRY 17	47 CS	24 CS

Table 3.3.5 Two-bit Full Adder

RESTRICTIONS / COST & TIME / FUNCTIONS (HEX)	FI = FO = FOX = FOO = 4, uncomplemented inputs only		
	TISLEV		JEFF (LEVLIM = 100)
	LEVLIM = 4	LEVLIM ≥ 5	
†			
1. SUM s_1	*		*
66996699	38089(4)	24053(5)	54129(5)
2. SUM s_0			
1E78E187	1585 CS	922 CS	467 CS
3. CARRY c_2			
01071F7F			

*It is not fan-in/fan-out restricted.

Table 3.3.6 Two-bit Full Adder

RESTRICTIONS / COST & TIME / FUNCTIONS (HEX)	FI = FO = FOX = FOO = 4, complemented and uncomplemented inputs		
	TISLEV		JEFF (LEVLIM = 100)
	LEVLIM = 3	LEVLIM ≥ 4	
†			
1. SUM s_1	*		*
66996699	33086(3)	24064(4)	48123(4)
2. SUM s_0			
1E78E187	1242 CS	912 CS	422 CS
3. CARRY c_2			
01071F7F			

†The addition is done as shown on the right.

$$\begin{array}{r} c_0 \\ a_1 a_0 \\ +\ b_1 b_0 \\ \hline c_2 s_1 s_0 \end{array}$$

Table 3.3.7 Two-bit Multiplier

RESTRICTIONS / COST & TIME / FUNCTIONS (HEX)	FI = FO = FOX = FOO = 3, uncomplemented inputs only		
	TISLEV		JEFF (LEVLIM = 100)
	LEVLIM = 4	LEVLIM ≥ 5	
1. 0505 2. 0356 3. 0032 4. 0001	* 13026(4) 139 CS	11021(5) 180 CS	26042(6) 126 CS

*It is not fan-in/fan-out restricted.

Table 3.3.8 Two-bit Multiplier

RESTRICTIONS / COST & TIME / FUNCTIONS (HEX)	FI = FO = FOX = FOO = 3, complemented and uncomplemented inputs		
	TISLEV		JEFF (LEVLIM = 100)
	LEVLIM = 3	LEVLIM ≥ 4	
1. 0505 2. 0356 3. 0032 4. 0001	* 9022(3) 110 CS	7017(4) 127 CS	18034(6) 102 CS

Table 3.3.9 Su-Nam's Example

RESTRICTIONS / FUNCTIONS (HEX) / COST & TIME	FI = FO = FOX = FOO = 2; complemented and uncomplemented inputs			
	TISLEV			JEFF
	LEVLIM = 4	LEVLIM = 5	LEVLIM $\geq$ 6	(LEVLIM =100)
Su-Nam's four-output functions	* 22040(4) 227 CS	* 20037(5) 194 CS	21037(6) 222 CS	27043(6) 139 CS

*It is not fan-in/fan-out restricted.

Functions used in this example (in truth form):

1. 000* 1*11 01*11111
2. 011* 1*11 00110000
3. 001* 1*00 001**000
4. 001* 1*01 01111111

(Su-Nam's result has 25 gates, 42 connections and 6 levels.)

are listed in the second, the third and the fourth columns. The last column gives the results derived by the transformation program JEFF (developed by J. G. Legge [15]) which can produce fan-in/fan-out restricted networks. Comparing the last two columns, we can find how effective the subroutine TISLEV is: in the case of 4-variable functions (Table 3.3.1), the networks derived by TISLEV have lower or equal costs and less and equal number of levels.* In the case of 5-variable functions (Table 3.3.2), three networks (function #4, #5 and #10) derived by TISLEV have higher costs and same number of levels. All other networks produced by TISLEV have fewer levels, although some have higher costs (function #2, #3, #6 and #9). Hence, for the single-output problems, subroutine TISLEV can usually give better initial networks from the viewpoint of the number of levels. The computation time is shown in the lower half part of these tables. TISLEV is observed to be comparable to JEFF in computation time. Comparing with the time spent by the transduction procedures (Chapter 5), TISLEV has reasonably good computation efficiency.

The results of several multiple-output examples are shown in Table 3.3.3 through Table 3.3.9 with different fan-in, fan-out and level restrictions. The results obtained by JEFF are also shown. Since the transduction procedure GTMERG (gate merge) is applied in TISLEV to simplify the network, the computation time is long. Table 3.3.3 and Table 3.3.4 give the initial networks for the one-bit full adder under different restrictions. TISLEV can always produce networks with lower costs and fewer levels. Table 3.3.5 and Table 3.3.6 give the results for the two-bit full adder. Apparently, the

*The number of levels is indicated by an integer which is parenthesized following the cost in the last two columns.

two-bit full adder is too complicated for JEFF to get a fan-in/fan-out restricted network. Table 3.3.7 and Table 3.3.8 are the results for the two-bit multiplier. Again, it is seen that TISLEV gives better results. Table 3.3.9 is the result of a four-output function which was used by Su-Nam [23] in testing their transformation programs. The network obtained by Su-Nam's algorithm has 25 gates, 42 connections and 6 levels. The network obtained by TISLEV has fewer gates and connections (21 gates and 37 connections) and the same number of levels.

All results for multiple-output network examples showed that TISLEV is more effective in finding a lower-level initial network, although the computation time is longer.

CHAPTER 4
MODIFICATION OF THE TRANSDUCTION PROCEDURES FOR LEVEL RESTRICTION

It is mentioned in Chapter 1 that the pruning procedures will not generate any new fan-in/fan-out problem or level problem since they only remove redundant gates or connections. The procedures which try to reconfigure the network by adding external variables or internal functions (i.e., the outputs of some gates) to some gates are modified for dealing with the level restriction. The following definitions are reviewed in order to facilitate the explanation in the following sections.

1. Permissible function: A function f is called a permissible function for a gate or a connection if after replacing the function realized at the gate or the connection by f, every output terminal of the network still realizes the same output function. The set of permissible functions of a gate v or a connection c is denoted by G(v) or G(c), respectively.
2. Maximum sets of permissible functions (MSPF): The set of all permissible functions for each gate v_i or connection c_{ij} is called an MSPF and is denoted by $G_M(v_i)$ or $G_M(c_{ij})$, respectively.
3. Compatible sets of permissible functions (CSPF): All sets $G(v_i)$ and $G(c_{ij})$ of permissible functions concerning a given network are said to be compatible if the following conditions are satisfied for all possible selections of subset U of gates and connections:
   (1) For each element u in U replace this element by an output from a network which realizes one function in G(u).

(2) After step (1), we get a new network. In this network any element w which is not contained in U realizes some function originally contained in G(w).

The CSPF of a gate v_i or a connection c_{ij} is denoted by $G_c(v_i)$ or $G_c(c_{ij})$, respectively.

4. Connectable functions: A function f is connectable to gate v_j with respect to $G_c(v_j)$ if and only if $f^{(d)} = 0$ holds for every d such that $G_c^{(d)}(v_j) = 1$.

5. Effectively connectable: A function f is effectively connectable to v_j with respect to $G_c(v_j)$ if (1) it is connectable to v_j with respect to $G_c(v_j)$ and (2) there exists at least one d satisfying the following condition:

$$f^{(d)} = 1, \quad G_c^{(d)}(v_j) = 0$$

6. Connectable (or effectively connectable) gates or external variables: A gate or an external variable v_i is connectable (or effectively connectable) to gate v_j if: (1) $f(v_i)$ is connectable (or effectively connectable) to gate v_j and (2) the network obtained after this input addition is loop-free.

7. Disconnectable connection: Any connection c_{ij} (connection between gate v_i and gate v_j, where v_i is an external variable or a gate) is said to be disconnectable from gate v_j with respect to set $G_c(v_j)$ if the output function of v_j remains in $G_c(v_j)$ after removing c_{ij} from the network.

8. Strongly effectively connectable: A gate or an external variable v_i is strongly connectable to gate v_j if: v_i is effectively connectable to v_j and (2) $IS(v_i) \supseteq IS(v_j)$ is not satisfied, where $IS(v_i)$ and $IS(v_j)$ are the sets of immediate successors of gate v_i and gate v_j, respectively.

9. Compatible set of permissible functions with errors (CSPFE): If the compatible set of permissible functions of a gate or an external variable v_i has some error-components due to the removal of some other gates or connections, then it is called a CSPFE and is denoted by $G_E(v_i)$.

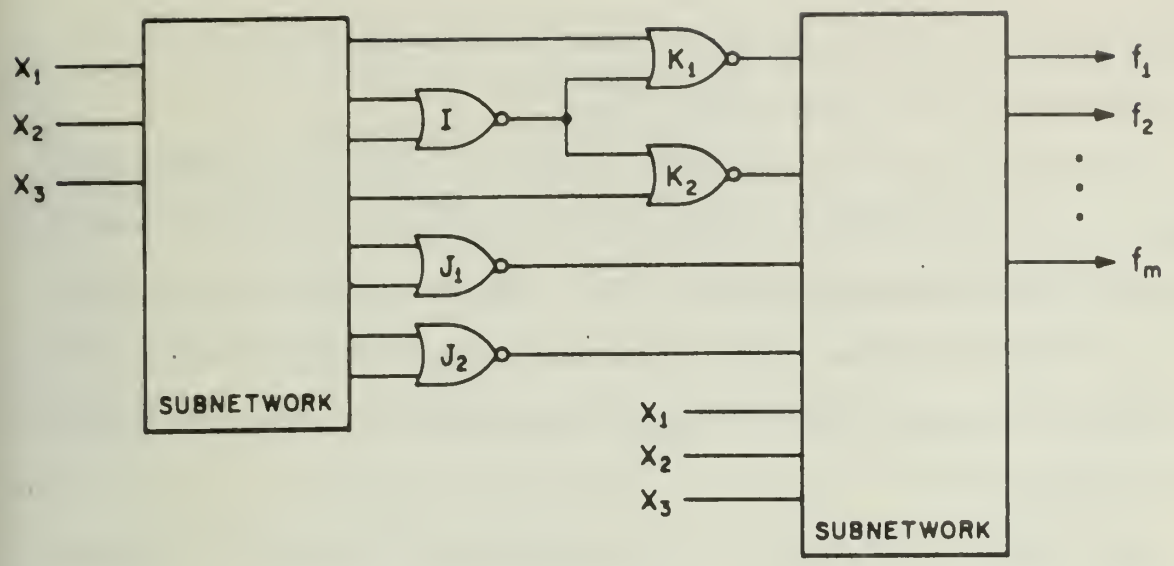
10. 0-error (or 0): A component of $G_c(v_i)$ is a 0-error if it is originally a 1 and it changes to zero because of the removal of other gates or connections.

11. 1-error (or 1): A component of $G_c(v_i)$ is a 1-error if it is originally a 0 and it changes to one because of the removal of other gates or connections.

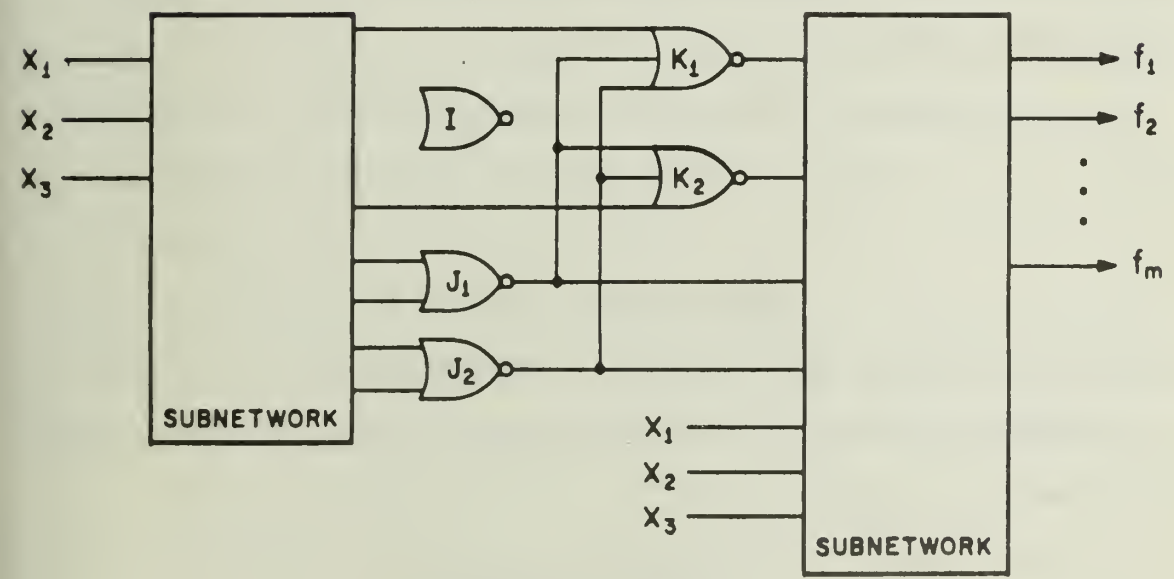
4.1 Procedure Based on Gate Substitution

The basic principle for gate substitution is to substitute the disjunction of the outputs of some gates and/or some external variables for the output of gate I. If this does not change the network outputs, gate I can be removed from the network. An example is shown in Figure 4.1.1. Assume the compatible sets of permissible functions in the network have already been calculated. Suppose that compatible sets of permissible functions of I, J_1, J_2, i.e., $G_c(I)$, $G_c(J_1)$ and $G_c(J_2)$ are (1**0110*), (*0*00101), and (1*10100*), respectively. It is easy to see that $G_c(I) \supseteq G_c(J_1) \vee G_c(J_2) = (1*101101)$. Therefore, the disjunction of the outputs of gates J_1 and J_2 can be used to substitute for the output of gate I and gate I can be removed. This is shown in Figure 4.1.1(b).

In the above example, no fan-in/fan-out or level restriction is considered. But if there is any restriction, then we have to check whether the substitution can be made, even if $G_c(I) \supseteq G_c(J_1) \vee G_c(J_2)$ holds. The



(a) Original network



(b) Resultant network

Fig. 4.1.1 Example of the transduction procedures based on gate substitution.

consideration of fan-in/fan-out restrictions is discussed in [22]. We are going to discuss the level restriction only.

In Figure 4.1.1(a), if the gate levels of J_1 and J_2 are greater than or equal to the gate level of I, then the number of levels of the network will not increase after the substitution is made. However, if the gate levels of either J_1 or J_2 is less than the gate level of I, the number of levels of the network may increase, i.e., the level restriction may be violated after the substitution.

For example, in Figure 4.1.1(a), assume the gate levels of I, J_1 and J_2 are 5, 3 and 5, respectively (indicated as GLEVEL(I) = 5, GLEVEL(J_1) = 3 and GLEVEL(J_2) = 5, respectively). Also assume that gate L is an immediate predecessor of gate J_1 and GLEVEL(L) = 4. After the substitution, GLEVEL(J_2) is still 5 but GLEVEL(J_1) becomes 5 and hence GLEVEL(L) becomes 6. If the number of levels of the original network is less than or equal to 5, then the substitution increases the number of levels of the network by 1.

Hence, in order not to violate the level restriction, the following check must be made in finding a gate J for substituting for the output of gate I.

Let J be a gate* for substituting for gate I, let DIST be the largest number of levels between J and L, where L is any predecessor of J (L is a gate or an external variable),† and let LEVLIM be the maximum number of levels which the network is allowed to have.

---

*If J is an external variable, then there will be no level problem to substitute J for I.

†The number of levels between J and the immediate predecessor L of J is 0 if L is an external variable.

If

$$GLEVEL(J) < GLEVEL(I) \quad \text{and}$$

$$GLEVEL(I) + DIST > LEVLIM \ ,$$

then J cannot be used to substitute for the output of gate I.

Subroutine SUBSTI [13,14,22] is the main control subroutine which implements the procedures based on gate substitution. The flowchart of subroutine SUBSTI before modification is shown in Figure 4.1.2. For level restriction, this is modified by inserting the flowchart in Figure 4.1.3 between block 6 and block 7 in Figure 4.1.2. In Figure 4.1.3, NOLEV is a subroutine which calculates the largest number of levels between J and L, where L is any predecessor of J (L is a gate or an external variable).

Figure 4.1.3 is explained in detail in the following:

Block 6.1: Check whether LEVLIM = 100 (this means that there is no level restriction) or J is an external variable. If either case is true, go to block 7 to check fan-in/fan-out restrictions (use of J as a candidate will not cause any level-restriction problem); otherwise, go to block 6.2.

Block 6.2: Check whether GLEVEL(J) < GLEVEL(I). If this is not true, go to block 7; otherwise, go to block 6.3.

Block 6.3: Call subroutine NOLEV(J) to calculate DIST which is the largest number of levels between J and L, where L is any predecessor of J. Go to block 6.4.

Block 6.4: Check whether GLEVEL(I) + DIST > LEVLIM. If this is true, J cannot be used (go to block 16); otherwise, go to block 7 to check fan-in/fan-out conditions.

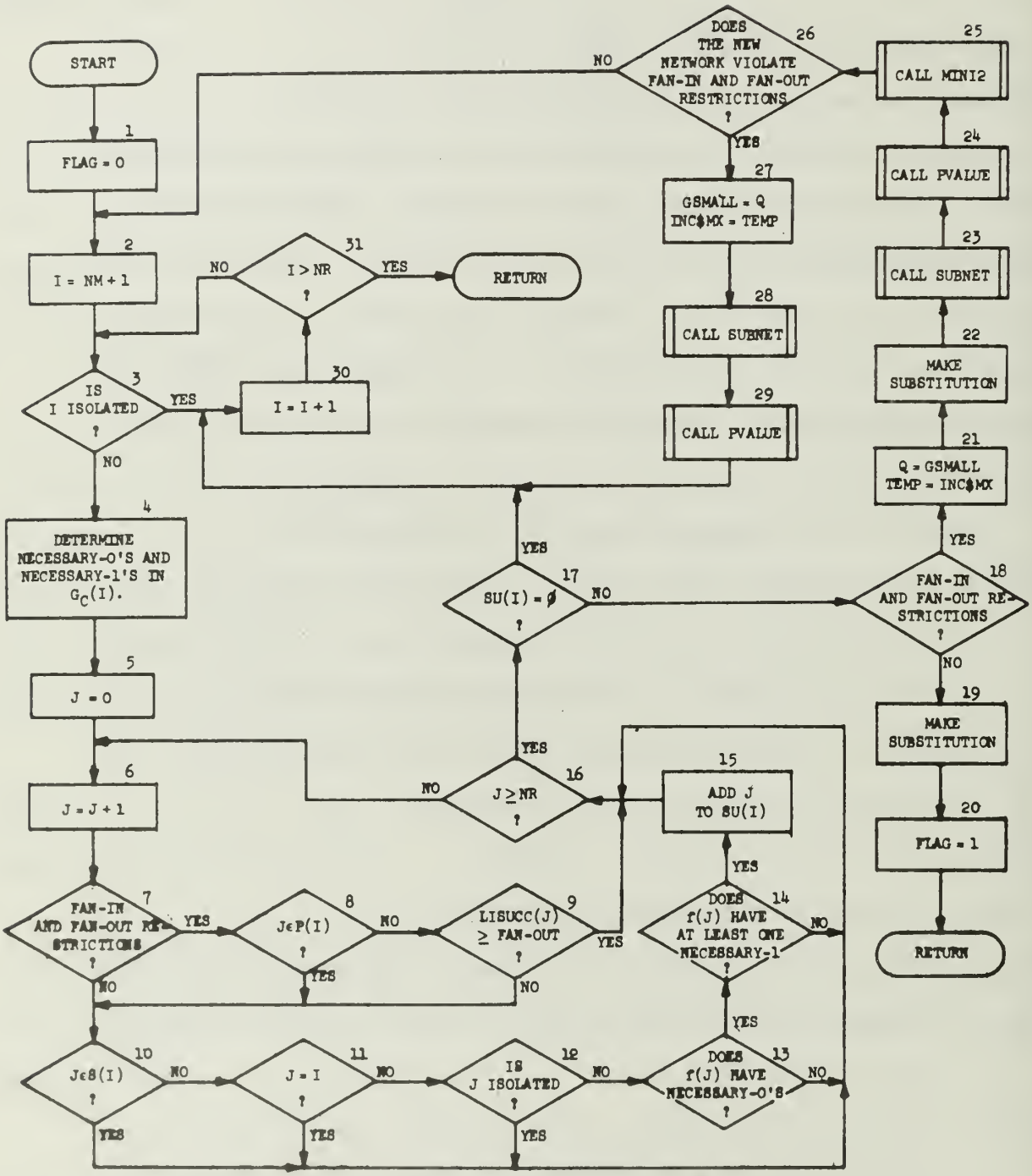
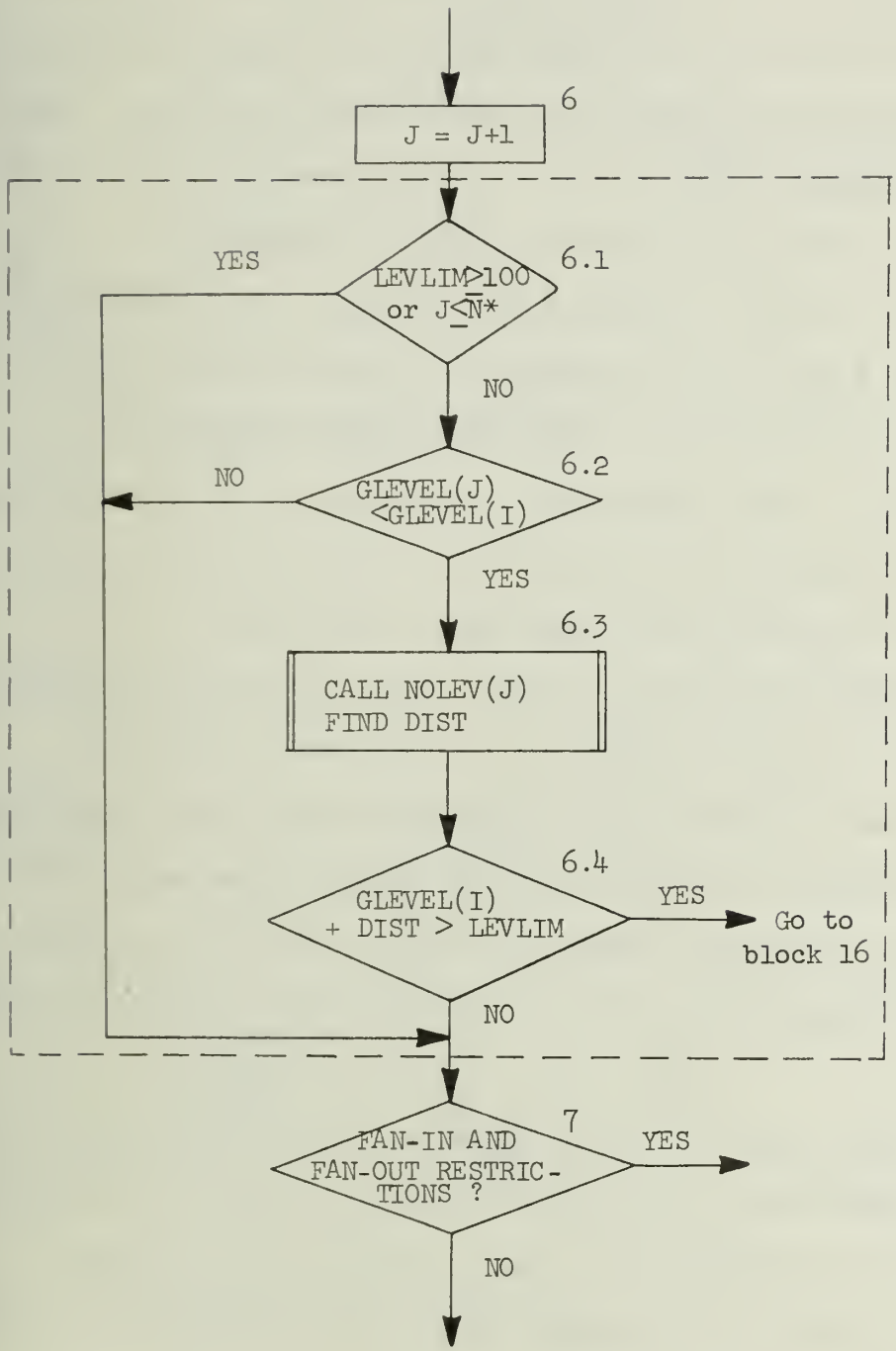


Figure 4.1.2 Flowchart of subroutine SUBSTI before modification for level restriction.



*N is the number of external variables. J is an external variable if J ≤ N or a gate if J > N (i.e., gates and external variables are numbered in this way).

Figure 4.1.3 Modification of subroutine SUBSTI for considering level restriction.

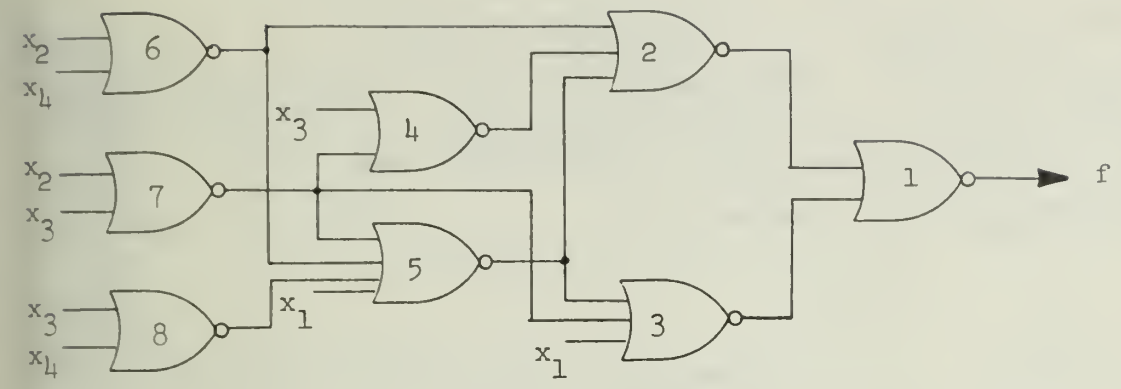
4.2 Procedures Based on Connectable and Disconnectable Functions

The basic idea in the transduction procedures based on connectable and disconnectable functions is to replace the input functions of gates by connectable functions and to remove disconnectable inputs from gates. Some gates may become redundant after the removal of disconnectable inputs. An example is shown in Figure 4.2.1, where the function realized is $f = \bar{x}_2\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2\bar{x}_4 \vee x_1x_2\bar{x}_3 \vee \bar{x}_1x_2x_4 \vee \bar{x}_1x_3x_4 \vee \bar{x}_1x_2x_3$. The network in Figure 4.2.1(a) has 8 gates and 20 connections. After calculating CSPF of the network, it is found that x_4 is connectable to gate 3 and gate 4 is connectable to gate 5. After making these connections, the connection between gate 8 and gate 5 is found to be disconnectable. The simplified network is shown in Figure 4.2.1(b) which has 7 gates and 19 connections.

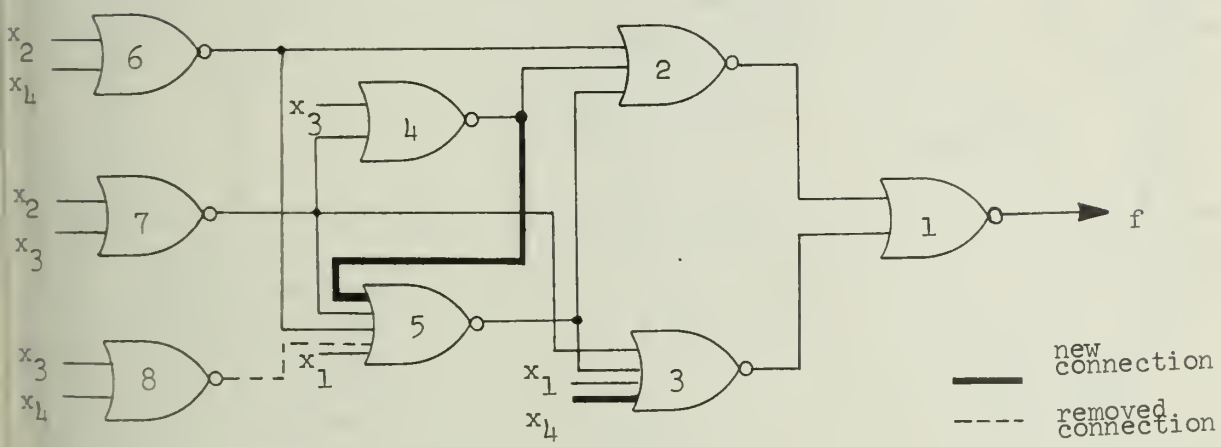
In this example, the number of levels of the network after transduction is increased by one. In the case that there exists level restriction, additional checks must be made in order not to violate the level restriction.

General flowcharts of subroutine PRIIFF, which is the central subroutine of transduction programs NETTRA-G1 and NETTRA-G2 [6,8], are shown in Figure 4.2.2 and Figure 4.2.3. The detailed explanation of these flowcharts can be found in [6].

In block 11 of Figure 4.2.3, all gates and external variables which are effectively connectable to gate GCO are listed according to the number of 0's covered. Now, for considering the level restriction, the following conditions must be checked to see whether those gates are really effectively connectable to GCO, i.e., whether any level problem will occur if they are connected to GCO:



(a) network before transduction



(b) network after transduction

Fig. 4.2.1 Example of the transduction procedures based on connectable and disconnectable functions.

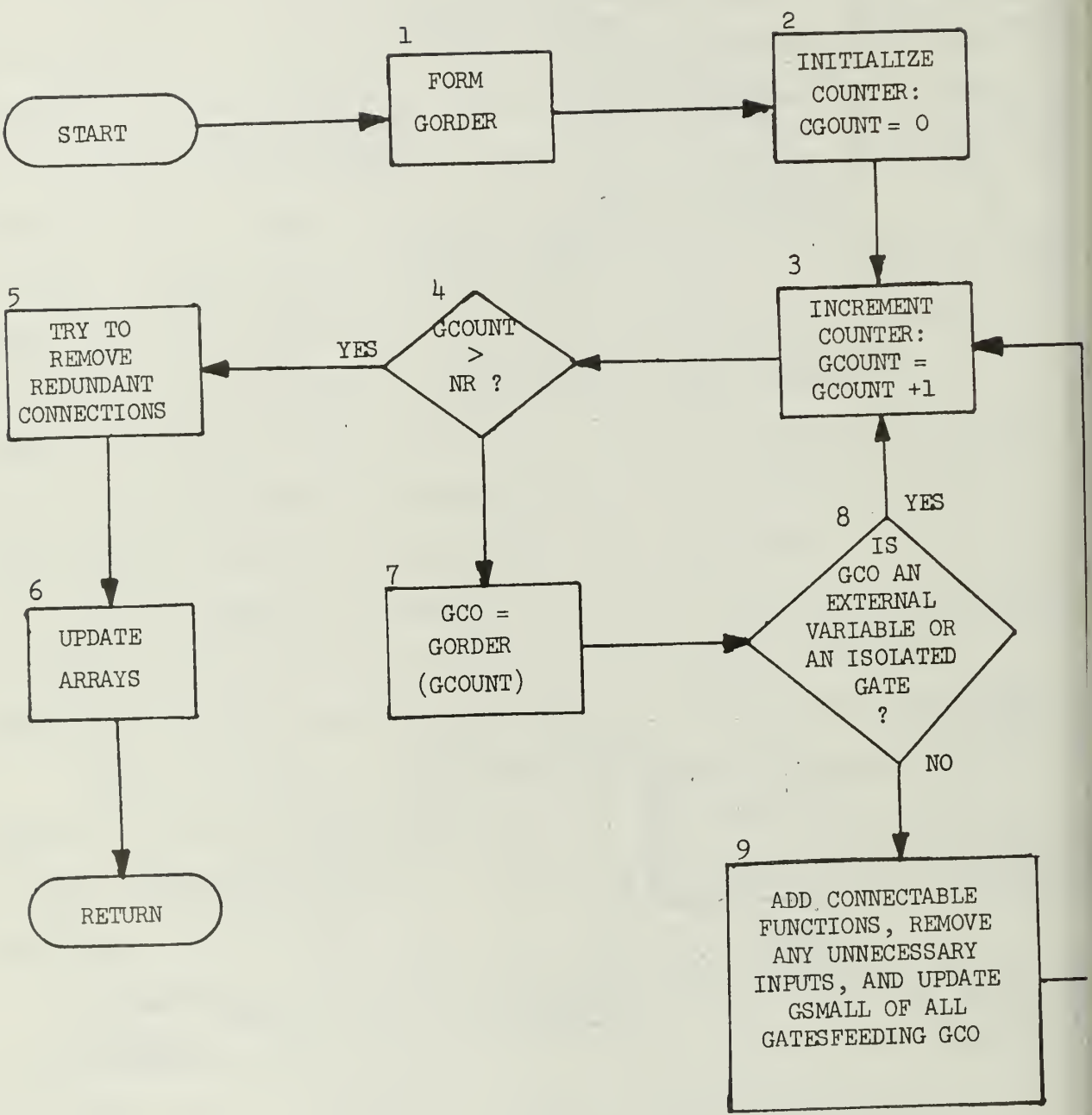


Figure 4.2.2 General flowchart of PRIIFF.

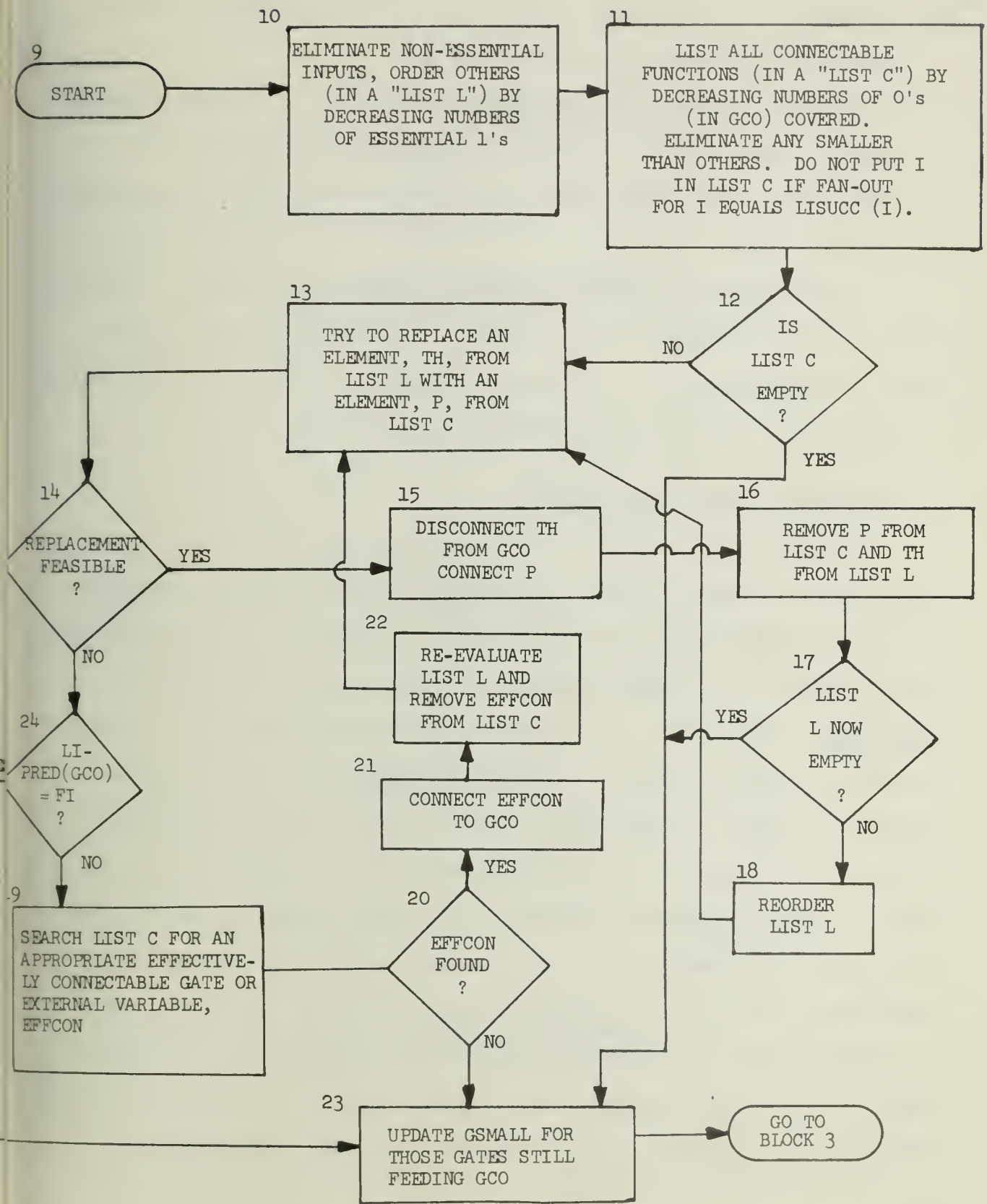


Figure 4.2.3 Details of block 9 of PRIIFF flowchart in Figure 4.2.2.

(1) If GLEVEL(I) > GLEVEL (GCO) then check the fan-out condition, where I is a gate (there will be no level problem).

(2) If GLEVEL(I) $\leq$ GLEVEL(GCO) and GLEVEL(GCO) + DIST + 1 > LEVLIM, then the output of gate I cannot be used as a connectable function for GCO, where DIST is the largest number of levels between I and any predecessor L of I.

The flowchart for the modification is shown on Figure 4.2.4. This is part of block 11 of Figure 4.2.3 and will be executed before checking the fan-out condition for I. The explanation of details of Figure 4.2.4 are omitted here since it is similar to that of Figure 4.1.3.

4.3 Procedures Based on Gate Merging

Two gates in a network are said to be mergeable if they can be replaced by one gate with inputs from external variables and/or existing gates which are not successors of these two gates without changing the function which the network realizes. The gate replacing two mergeable gates is called the merged gate. The transduction procedures based on gate merging try to merge two gates at a time in order to simplify the network. An example for these procedures is shown in Figure 4.3.1. Suppose gates I and J are mergeable into gate IJ with inputs from gates K, M and N which are not successors of gate I or gate J, as shown in Figure 4.3.1(a). The transduction procedures will construct the merged gate IJ with inputs from gates K, M and N, and remove gates I and J from the network. They, then, will connect the output of the merged gate IJ to all immediate successors of gates I and J as shown in Figure 4.3.1(b). The resultant network still realizes the given output function (see [11,13,22] for details of the procedures and their principles).

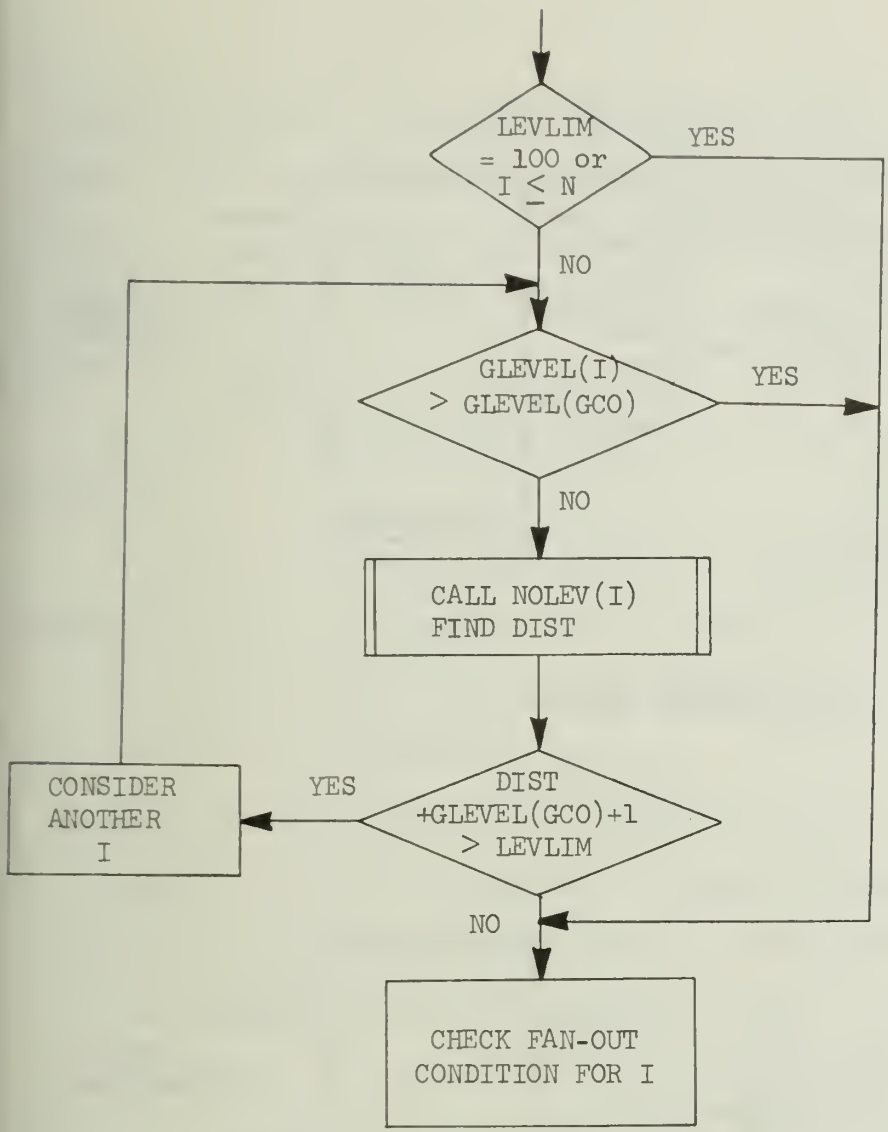
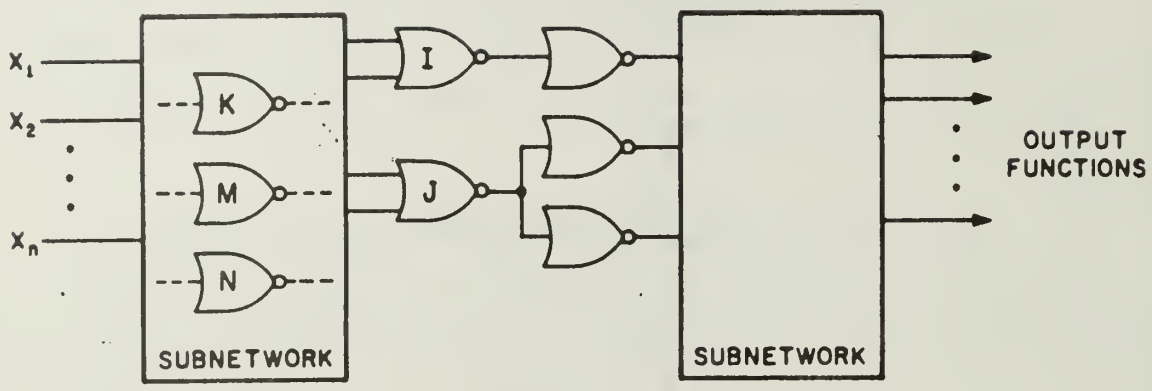
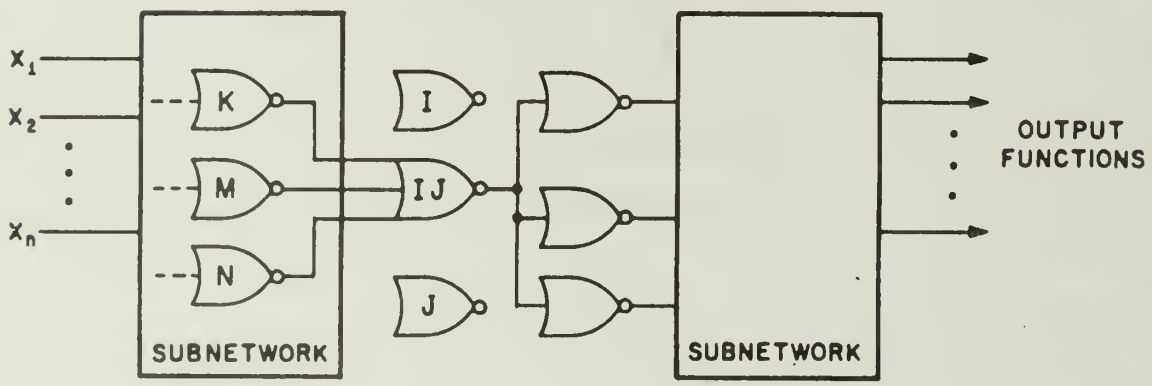


Figure 4.2.4 Flowchart of the modification of block 11 of Figure 4.2.3 for level restriction.



(a) Original network



(b) Resultant network

Fig. 4.3.1 Example of the transduction procedures based on gate merging.

The flowchart of the central subroutine GTMERG of the transduction procedures based on gate merging is shown in Figure 4.3.2. Detailed explanation of this flowchart can be found in [22]. In Figure 4.3.2, the substituability of gate I for gate J (or gate J for gate I) is checked in blocks 9, 10 and 11:*

If $f(I) \in G_c(J)$ and $I \notin S(J)$, gate I can substitute for gate J. Go to block 29.

If $f(J) \in G_c(I)$ and $J \notin S(I)$, gate J can substitute for gate I. Interchange the labels of I and J and go to block 29.

In case that level restriction is considered, the following checks must be made before executing block 29:

(1) If GLEVEL(I) $\geq$ GLEVEL(J), then there will be no level problem.

(2) If GLEVEL(I) < GLEVEL(J), then gate I cannot substitute for gate J if GLEVEL(J) + DIST > LEVLIM, where DIST is the largest number of levels between I and any predecessor L of I.

The flowchart of this modification is shown in Figure 4.3.3. Again the explanation is omitted.

In Figure 4.3.2, block 12 through block 22 try to find inputs for the merged gate IJ.

---

*The substitution of one gate for another gate is a special case of the substitution discussed in section 4.1.

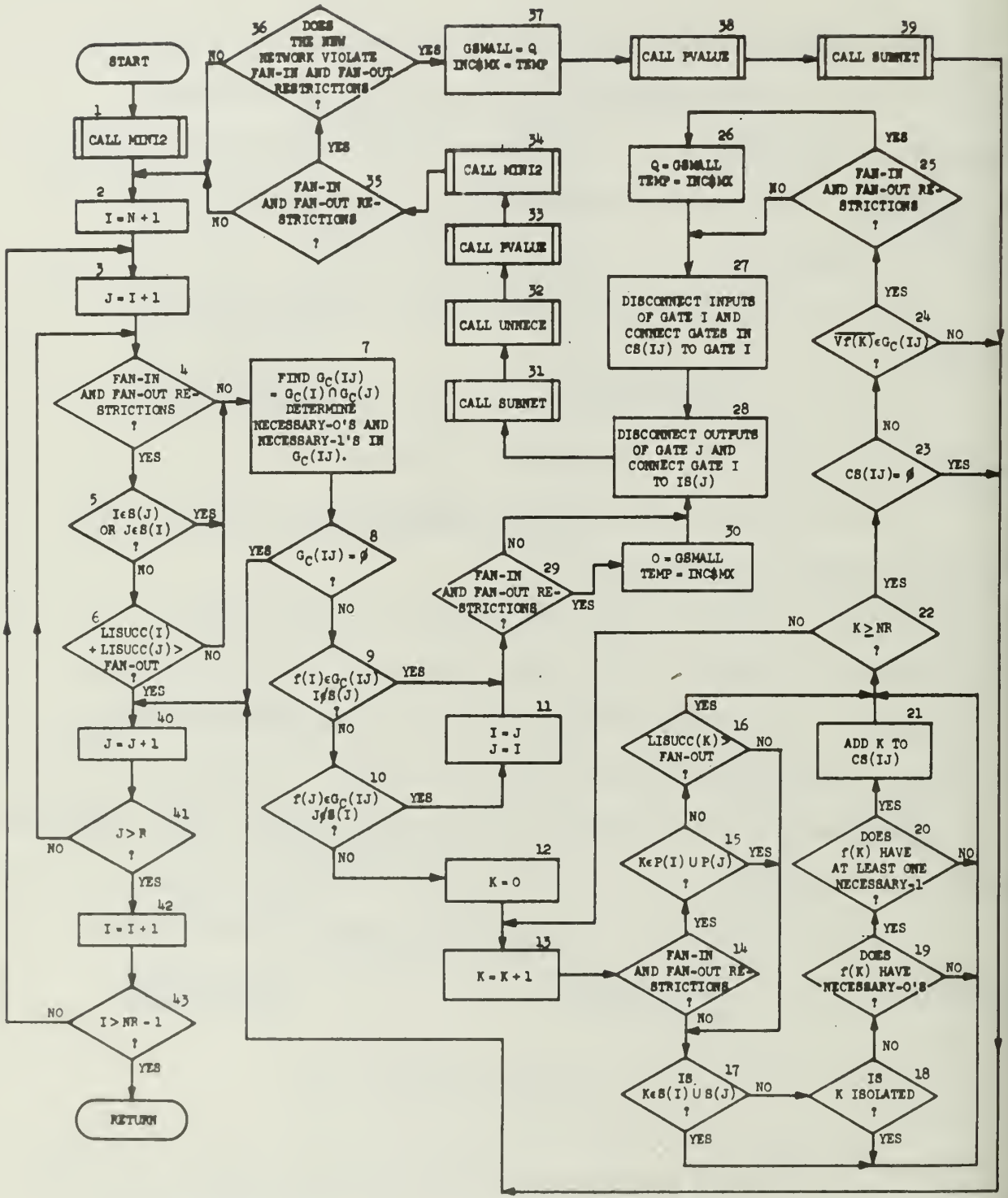


Figure 4.3.2 Flowchart of modified subroutine GTMERG.

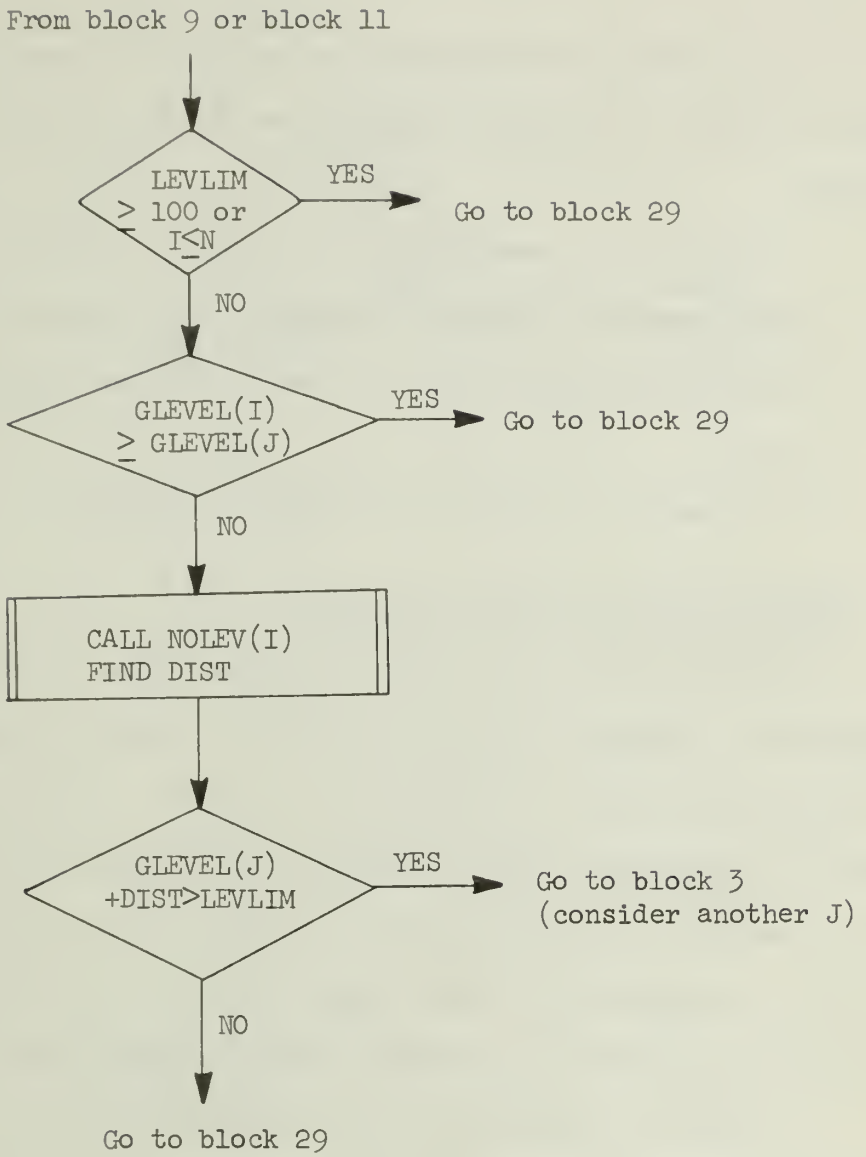


Figure 4.3.3 Modification for level restriction. These blocks must be inserted after block 9 and block 11 but before block 29 in Figure 4.3.2.

The following conditions must be satisfied by gate K whose output is going to be used to feed gate IJ (i.e., use the output of K as an input of gate LJ):

If DIST + 1 + max(GLEVEL(I), GLEVEL(J)) > LEVLIM, then gate K cannot be used as an input of gate IJ, where DIST is the largest number of levels between K and any predecessor L of K.

The flowchart in Figure 4.3.4 shows the detail.

4.4 Procedures Based on Error-compensation

The basic idea in the transduction procedures based on error-compensation is simple. In any given network, a gate is selected according to an appropriate order. Assuming that this gate is removed, check the outputs of the network. If the outputs do not change, then the selected gate is redundant and can be actually removed. If the outputs do change, then try to compensate for these changes (errors) by reconfiguring the network. The selected gate becomes redundant if those changes can be compensated for.

In order to compensate for error-components in the CSPFE of a gate, functions currently realized at other gates and external variables may be used [10]. In order to make the error-compensation more flexible, the concepts of potential outputs and potential output tables (POT) were introduced in [10], and a procedure utilizing a potential output table was also given. A potential output from a gate I is a function realized at I by connecting additional inputs to I. This potential output of I usually differs from the function currently realized at I, and therefore can be used to compensate for errors

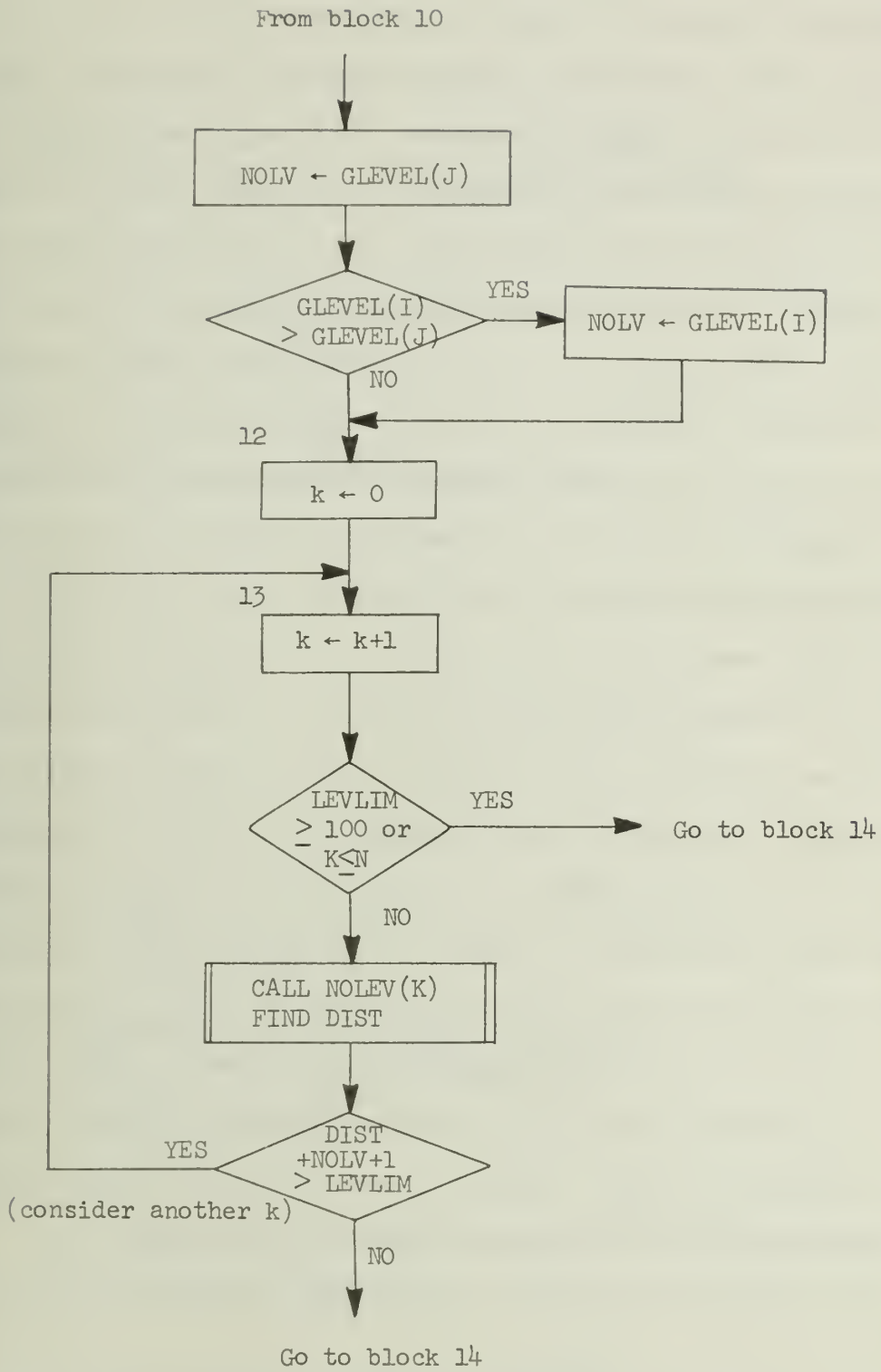


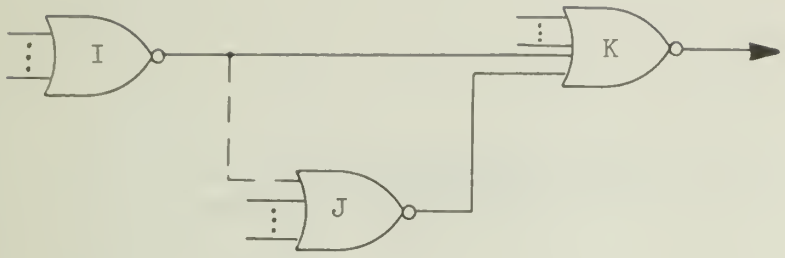
Figure 4.3.4 Flowchart of modification for level restriction between block 10 and block 14 in Figure 4.3.2.

in a certain gate to which the current output function at I is not connectable. The principle used in generating potential output for a given NOR (NAND) network is called the triangular condition. In a loop-free NOR (NAND) network, suppose three gates are connected to each other to form a triangle. Then the connection from the highest level gate to the second highest level gate is redundant for realizing the output of the lowest level gate if the second highest level gate has no output connection other than the one to the lowest level gate. In Figure 4.4.1(a), the dashed connection is redundant. Conversely, if two gates are both immediate predecessors of another gate, adding a connection between the first two will not affect the output of the other gate if the gate to which the new connection feeds has no output connection other than the one to the third gate. In Figure 4.4.1(b), gate I and gate J are immediate predecessors of gate K. The connection of gate I to gate J will not change the output of gate K. But the output of gate J after making the bold-line connection may be different from that before making the connection. In a similar manner, if three gates are all immediate predecessors of another gate, then adding two outputs to the third gate will not affect the output of the lowest level gate if the third gate has no output connection other than the one to the lowest level gate. In Figure 4.4.2, the networks in (a), (b), (c) and (d) realize the same function, but the outputs of gate K are not necessarily the same. Therefore, these different outputs of gate J can be used to compensate for error-components for other gates.

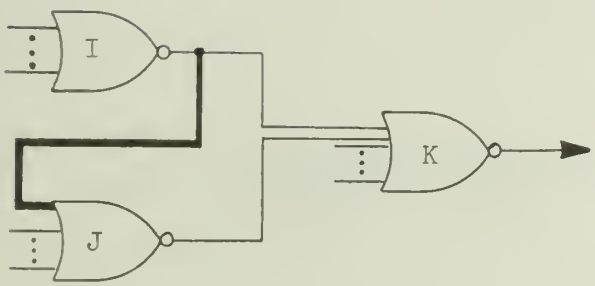
The transduction procedures based on error-compensation [10] are as follows:

Step 1: Select a gate according to an appropriate order.* If all gates have been considered, then stop.

*Usually according to the number of 0-components a gate has.

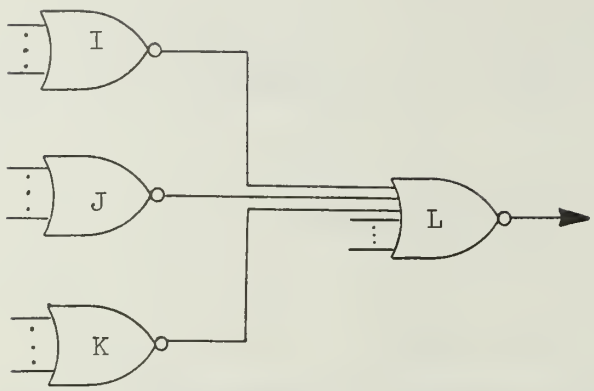


(a) The dashed connection is redundant.

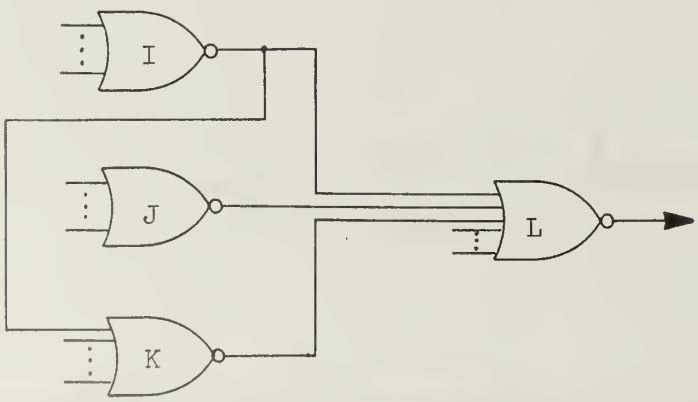


(b) The bold-line connection can be made without changing the output of gate K.

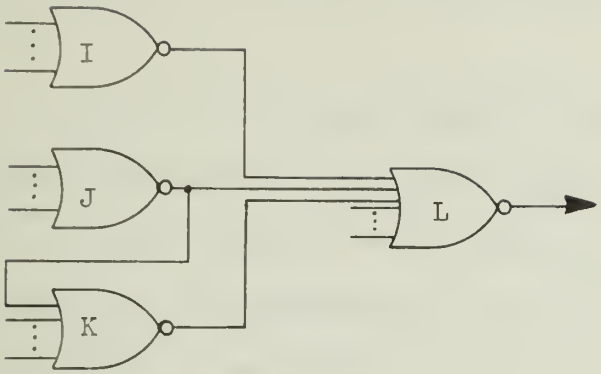
Fig. 4.4.1 Triangular condition



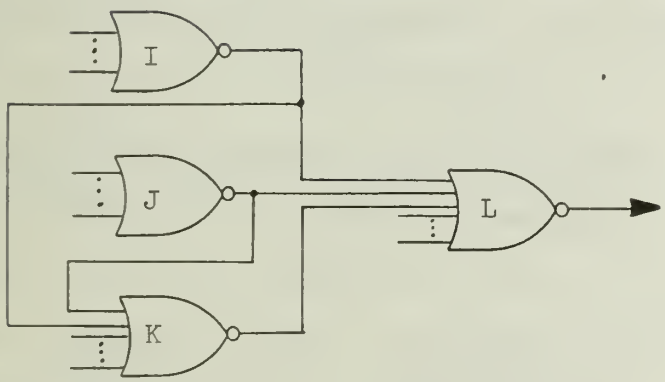
(a) the original network



(b) connecting I to K



(c) connecting J to K



(d) connecting I and J to K

Fig. 4.4.2 A more general example of triangular condition.

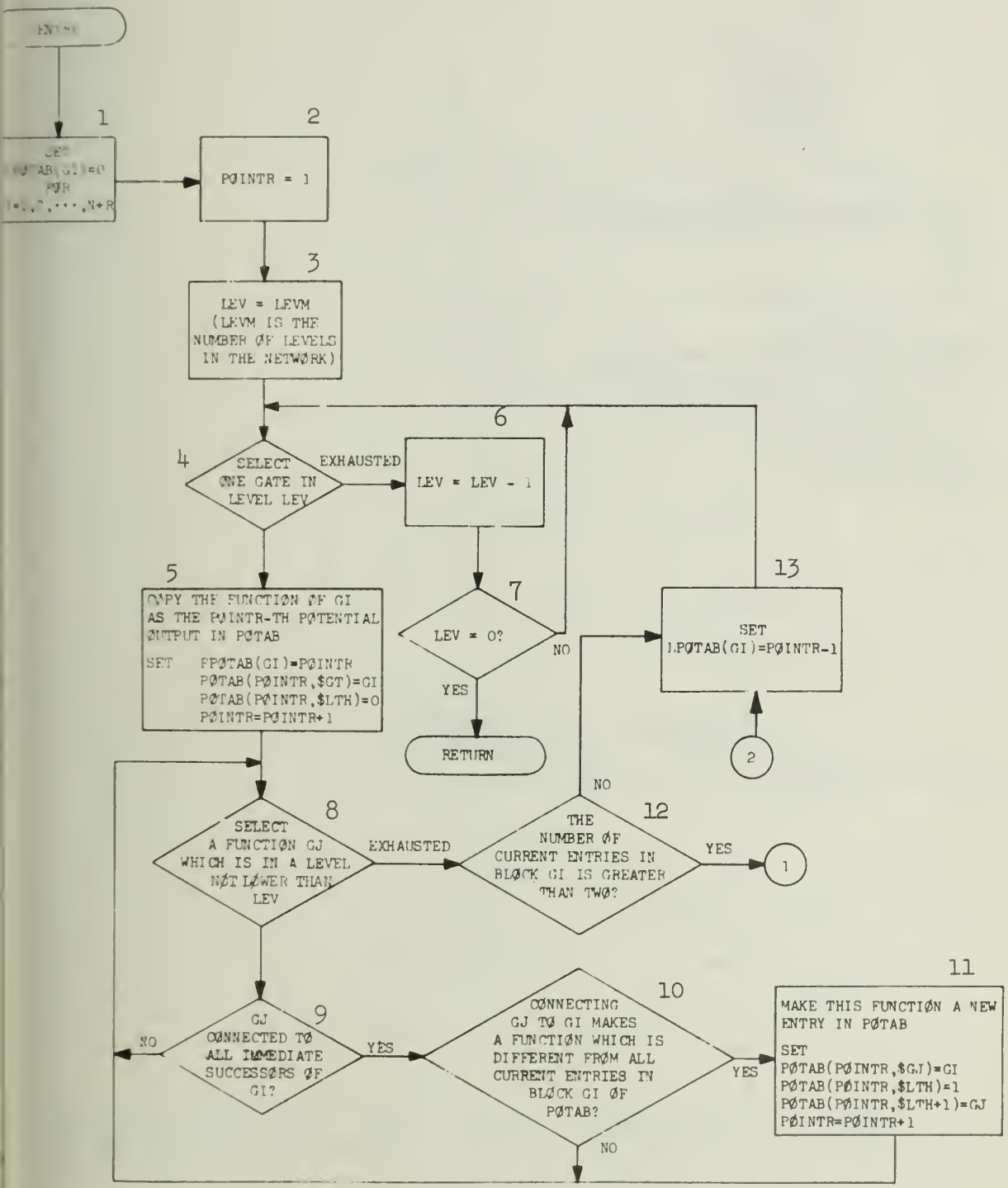
Step 2: Assume the selected gate is removed. Check whether the outputs of the network change or not. If the outputs do not change, then actually remove this gate and go to step 1. Otherwise, go to next step.

Step 3: Construct the potential output table, go to step 4.

Step 4: Starting from the output gates, try to find some potential output functions which can compensate for the error-components found in step 2. If the errors can be compensated for, then remove the selected gate and go to step 1. Otherwise, propagate the errors to the next higher level and repeat this step. If the errors have already been propagated to the highest level and still cannot be compensated for, then go to step 1.

These four steps are mainly realized by eight subroutines: PROCCE, RCEC, POT, RPLCF, CALS1, CONECT, ORDRQ2 and FORC [12]. Among them, subroutines POT and RCEC are modified for level restrictions. Subroutine POT constructs the potential output table. Figure 4.4.3 gives the flowchart of this subroutine When there exists level restriction, each entry (potential output function) in the potential output table must not generate any level problem if it is used to compensate for errors. Assume I is a gate from which a new potential output function can be formed if gate J is connected to I. If GLEVEL(I) $<$ GLEVEL(J), then there will be no any level problem to make this potential output. If GLEVEL(I) $\geq$ GLEVEL(J), then gate J cannot be connected to gate I if GLEVEL(I) + DIST + 1 $>$ LEVLIM, where DIST is the largest number of levels between J and any predecessor L of J. These checks are made between block 8 and block 9 of Figure 4.4.3.* The modified part is shown in Figure 4.4.. Subroutine RCEC (Replacement of Connections for Error-Compensation) is the main part of error-compensation procedures. When it is called by subroutine

*The modification for fan-in/fan-out restrictions is not shown in Figure 4.4.3. Details can be found in [7].


1
2
PØINTR = 1
3
LEV = LEVM
(LEVM IS THE
NUMBER ØF LEVELS
IN THE NETWØRK)
6
4
SELECT
ØNE GATE IN
LEVEL LEV
EXHAUSTED
LEV = LEV - 1
5
CØPY THE FUNCTIØN ØF GI
AS THE PØINTR-TH PØTENTIAL
ØUTPUT IN PØTAB
SET PPØTAB(GI)=PØINTR
PØTAB(PØINTR,$GT)=GI
PØTAB(PØINTR,$LTH)=0
PØINTR=PØINTR+1
7
LEV = 0?
NO
YES
RETURN
13
SET
PØTAB(GI)=PØINTR-1
2
NO
8
SELECT
A FUNCTIØN GJ
WHICH IS IN A LEVEL
NØT LØWER THAN
LEV
EXHAUSTED
12
THE
NUMBER ØF
CURRENT ENTRIES IN
BLØCK GI IS GREATER
THAN TWØ?
YES
1
11
MAKE THIS FUNCTIØN A NEW
ENTRY IN PØTAB
SET
PØTAB(PØINTR,$GJ)=GI
PØTAB(PØINTR,$LTH)=1
PØTAB(PØINTR,$LTH+1)=GJ
PØINTR=PØINTR+1
10
CØNNECTING
GJ TØ GI MAKES
A FUNCTIØN WHICH IS
DIFFERENT FRØM ALL
CURRENT ENTRIES IN
BLØCK GI ØF
PØTAB?
YES
NO
9
GJ
CØNNECTED TØ
ALL IMMEDIATE
SUCCESSØRS ØF
GI?
NO
YES

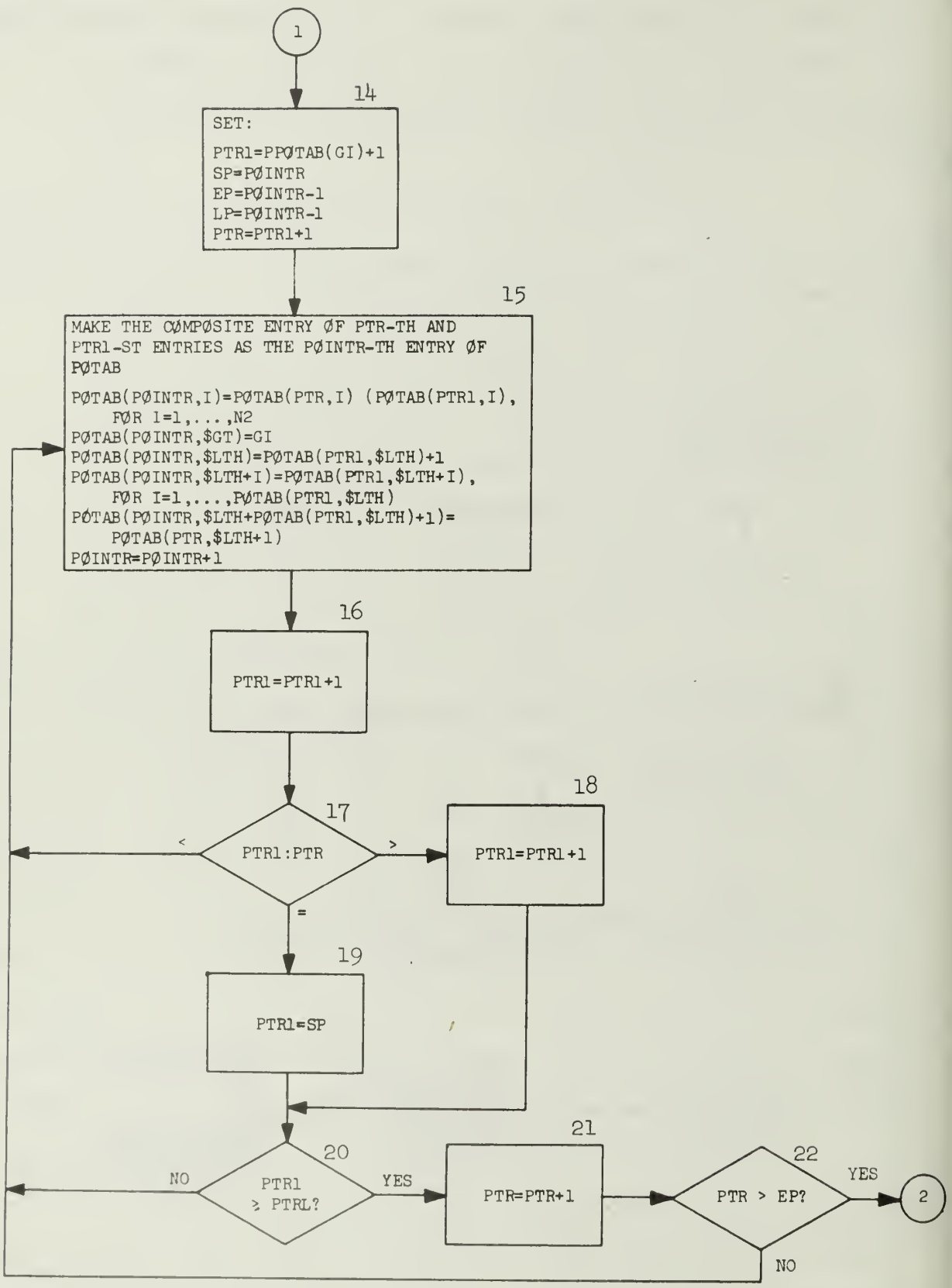


Figure 4.4.3 Flowchart of subroutine POT.

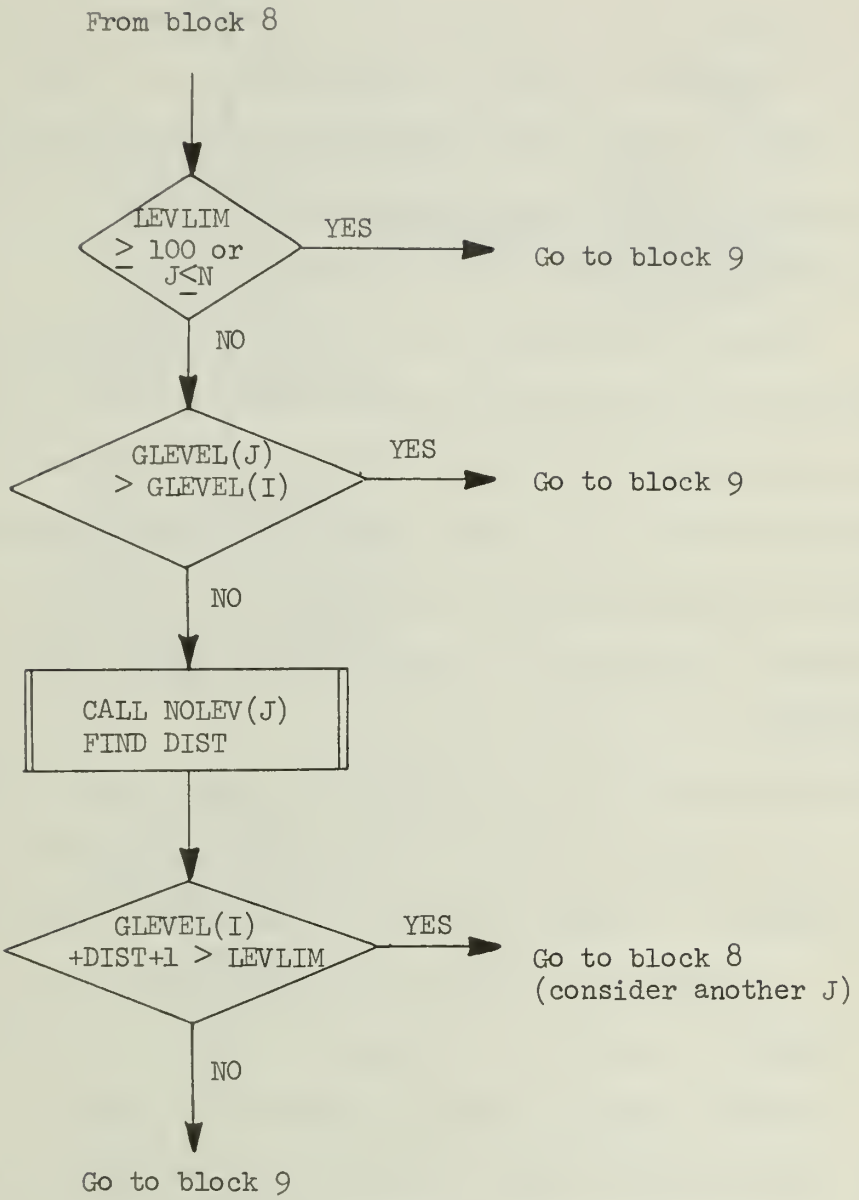


Figure 4.4.4 Modification in POT for level restriction.

PROCCE, the CSPFE of a given NOR network which does not realize the given output functions will be calculated. It then selects one gate at a time and tries to compensate for the error-components in the CSPFE of the selected gate. Detailed steps of this subroutine is fairly complicated. For the sake of convenience, the original flowchart [12] of RCEC is shown in Figure 4.4.5; but only the modification will be explained.

In step 3 of Figure 4.4.5, it selects from potential output table all effectively E-connectable functions (ECF) for gate GI which has some error-components in the output function. Each of ECF covers at least one component in CO or C1E.* Again each of ECF must not generate any level problem if level restriction is considered. The following checks are made to guarantee that there is no level problem after the error-components are compensated:

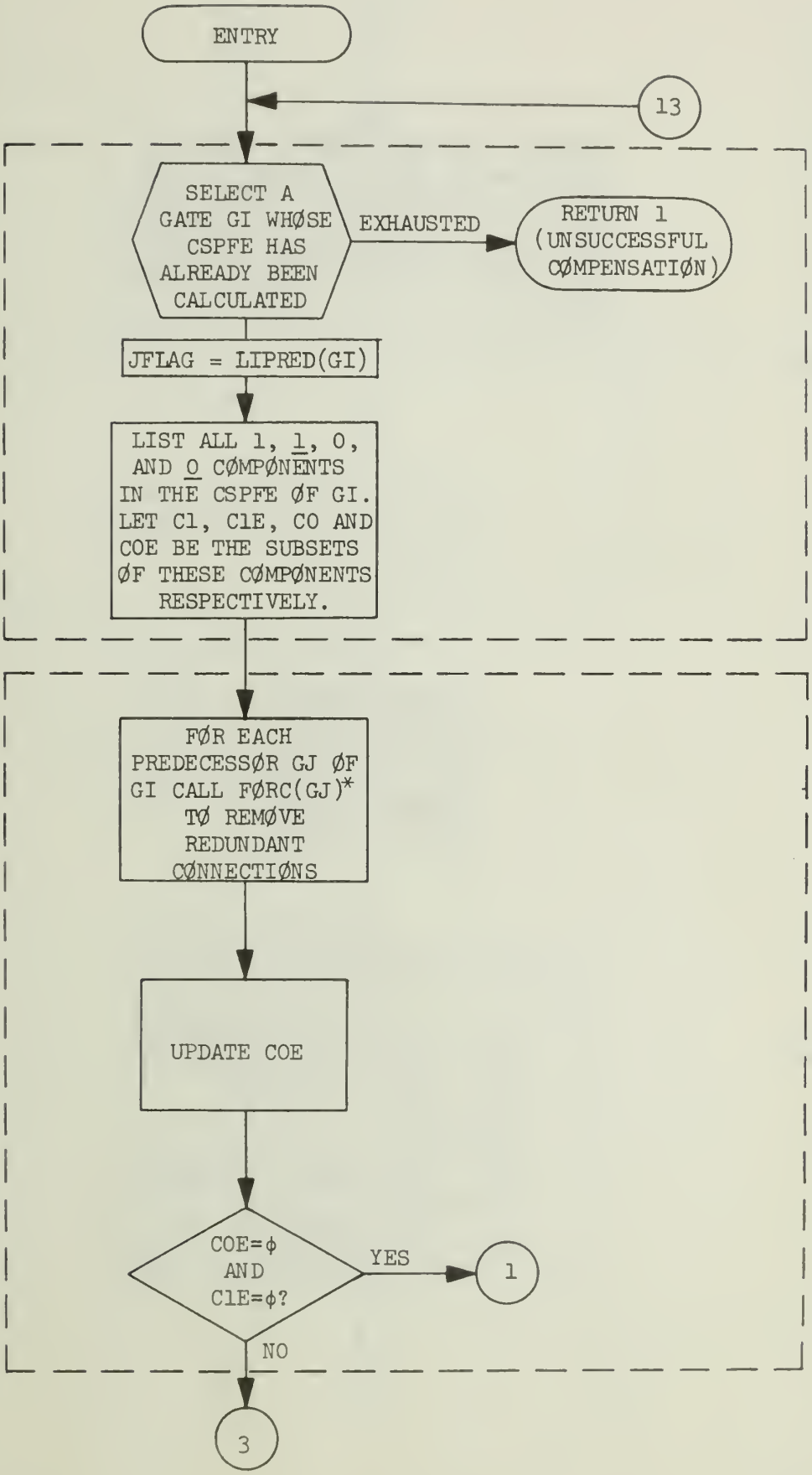
(1) Assume gate GJ is one of ECF for GI and the function realized at GJ is the original output of GJ, i.e., not a potential output. If GLEVEL(GJ) > GLEVEL(GI), then there will be no level problem.

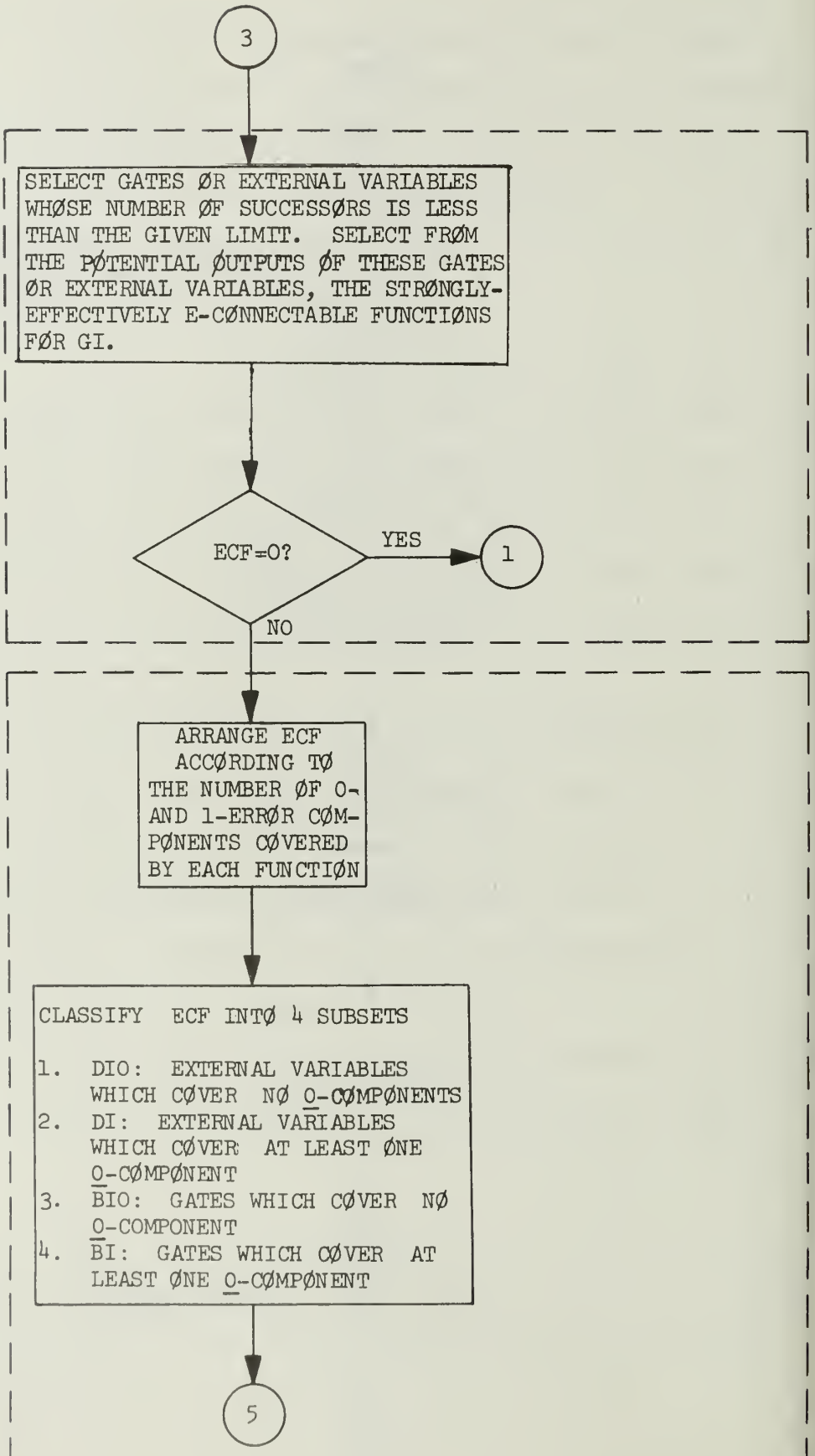
    If GLEVEL(GJ) $\leq$ GLEVEL(GI), then gate GJ cannot be used if GLEVEL(GI) + DIST + 1 > LEVLIM, where DIST is the largest number of levels between GJ and any predecessor GL of GJ.

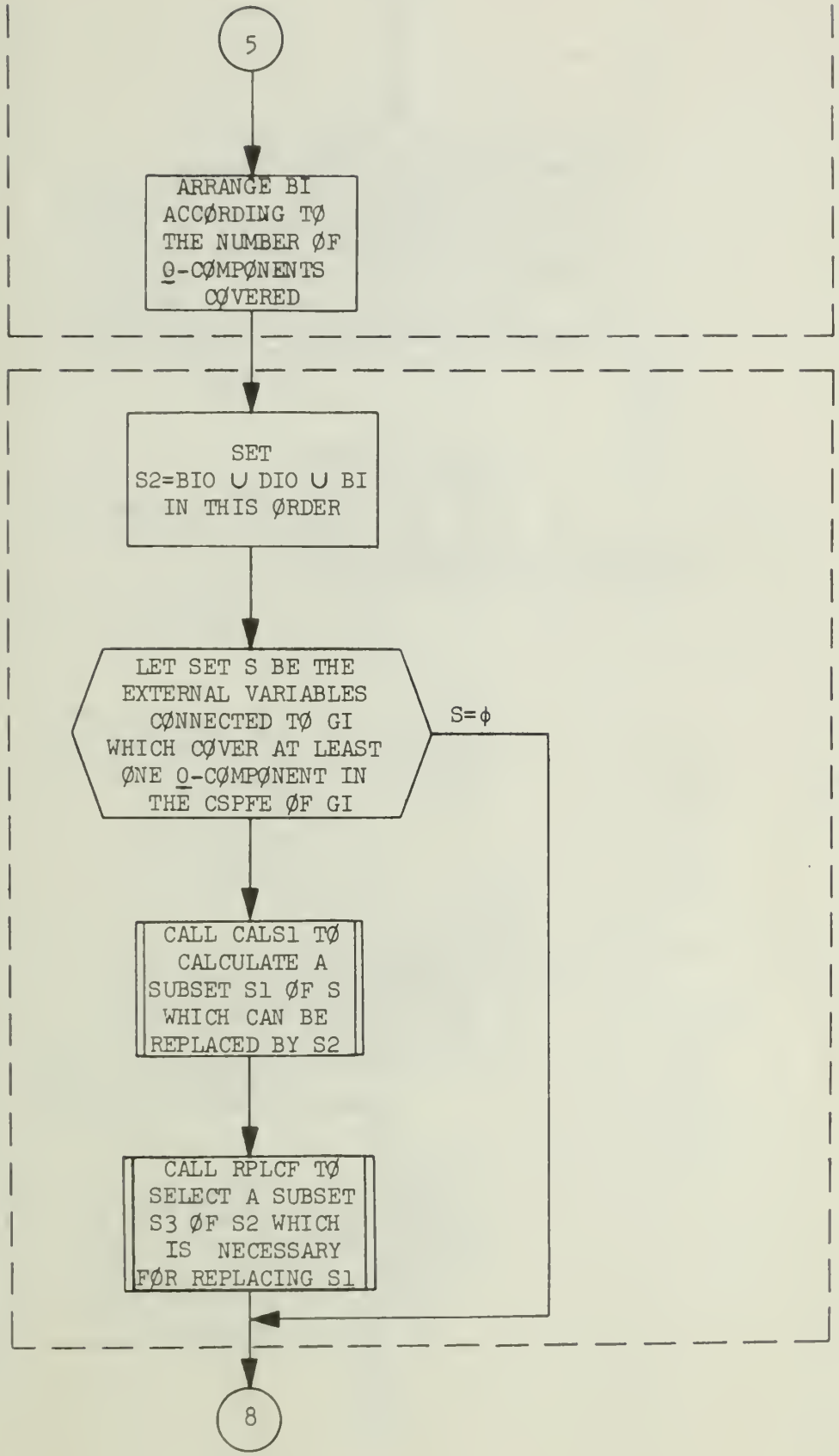
(2) Assume gate GJ is one of ECF for GI and the function realized at GJ is one of the potential outputs of GJ, i.e., not the original output of GJ.

---

*CO and C1E are the sets of 0-components and 1-error-components of gate GI, respectively.

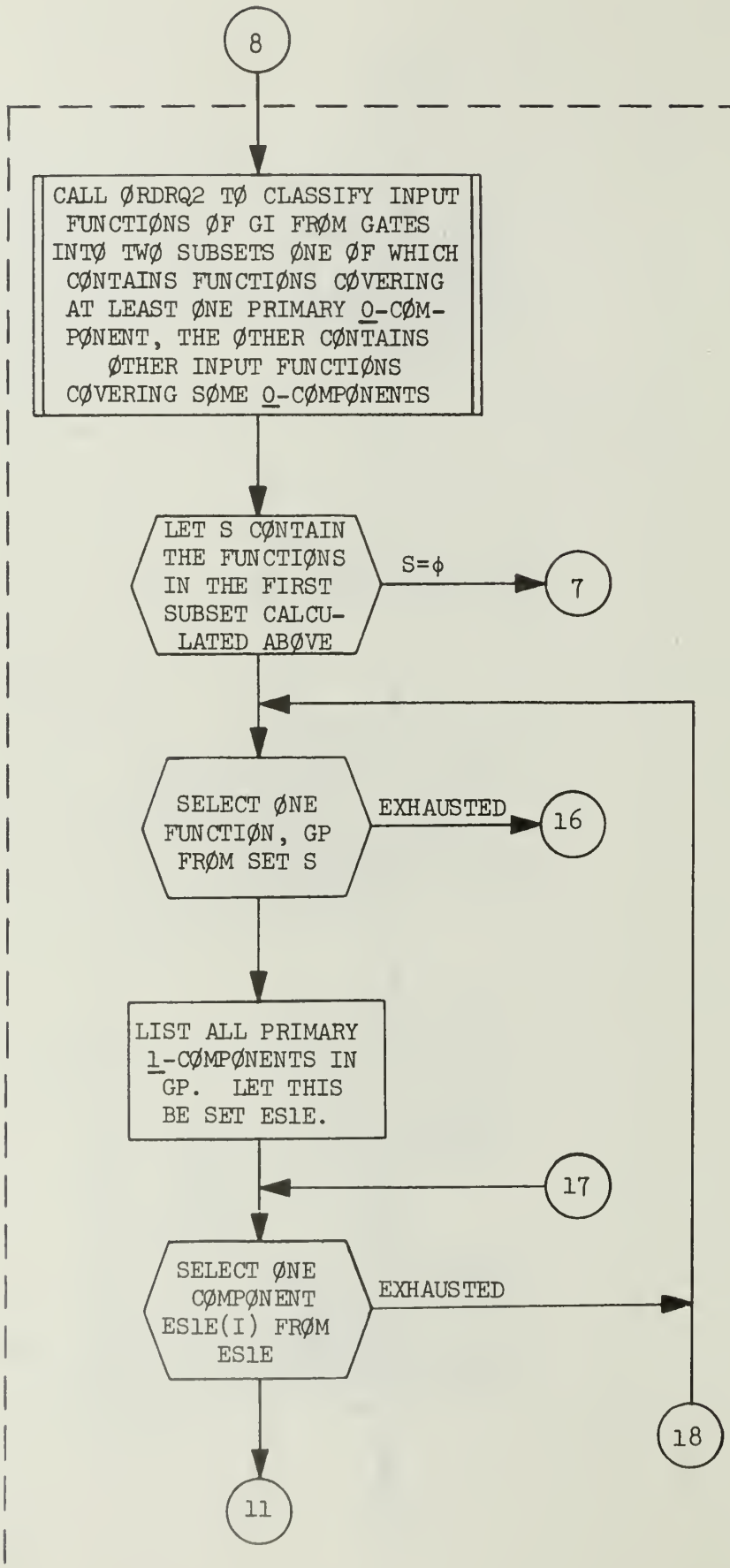

ENTRY
13
SELECT A GATE GI WHØSE CSPFE HAS ALREADY BEEN CALCULATED
EXHAUSTED
RETURN 1 (UNSUCCESSFUL CØMPENSATIØN)
JFLAG = LIPRED(GI)
Step 1
LIST ALL 1, 1, 0, AND 0 CØMPØNENTS IN THE CSPFE ØF GI. LET C1, C1E, CO AND COE BE THE SUBSETS ØF THESE CØMPØNENTS RESPECTIVELY.
FØR EACH PREDECESSØR GJ ØF GI CALL FØRC(GJ)* TØ REMØVE REDUNDANT CØNNECTIØNS
Step 2
UPDATE COE
COE=ϕ AND C1E=ϕ?
YES
1
NO
3

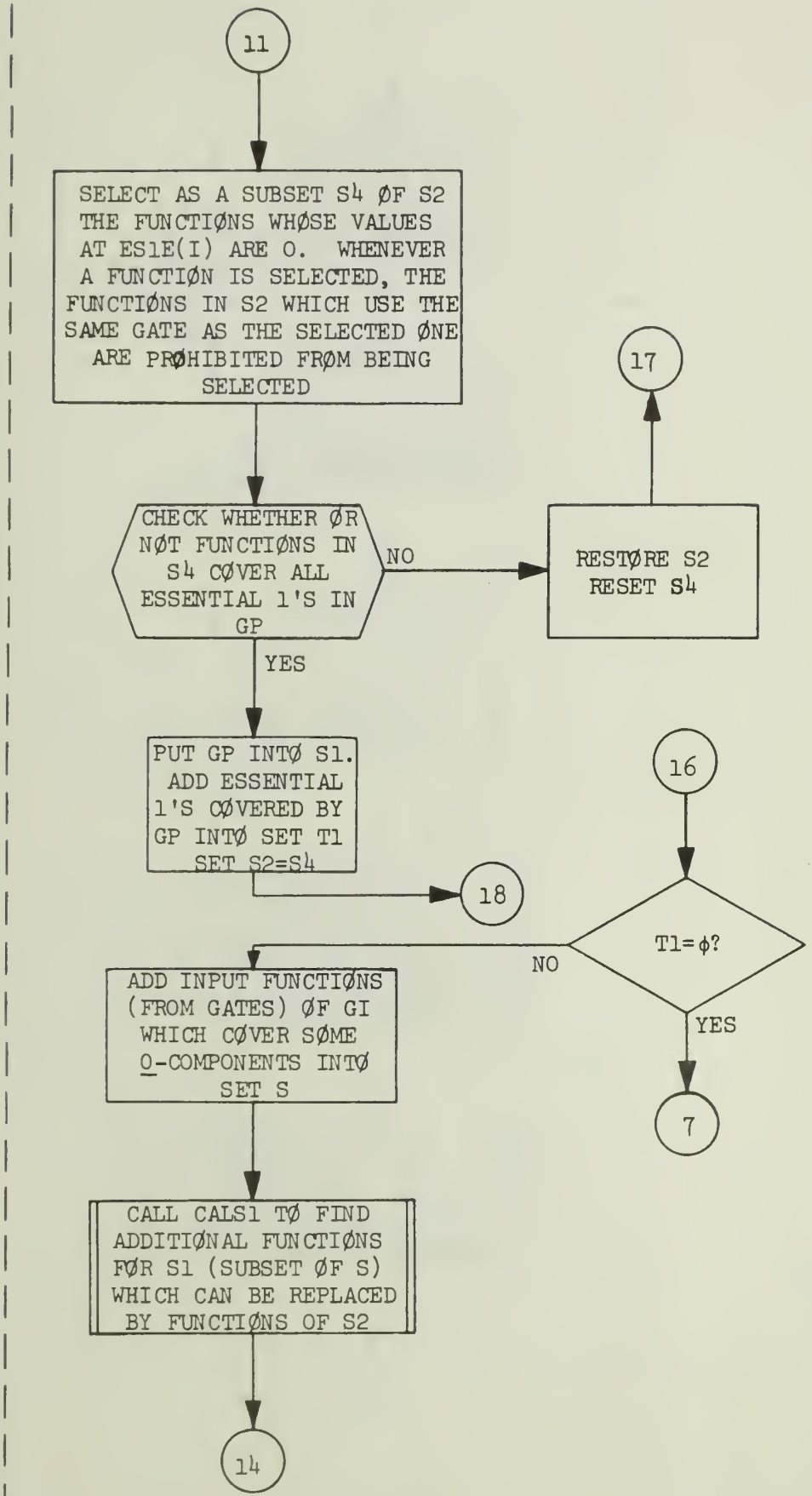

3
SELECT GATES ØR EXTERNAL VARIABLES WHØSE NUMBER ØF SUCCESSØRS IS LESS THAN THE GIVEN LIMIT. SELECT FRØM THE PØTENTIAL ØUTPUTS ØF THESE GATES ØR EXTERNAL VARIABLES, THE STRØNGLY-EFFECTIVELY E-CØNNECTABLE FUNCTIØNS FØR GI.
Step 3
ECF=0?
YES
1
NO
ARRANGE ECF ACCØRDING TØ THE NUMBER ØF 0- AND 1-ERRØR CØM-PØNENTS CØVERED BY EACH FUNCTIØN
CLASSIFY ECF INTØ 4 SUBSETS
1. DIO: EXTERNAL VARIABLES WHICH CØVER NØ 0-CØMPØNENTS
2. DI: EXTERNAL VARIABLES WHICH CØVER AT LEAST ØNE 0-CØMPØNENT
3. BIO: GATES WHICH CØVER NØ 0-COMPONENT
4. BI: GATES WHICH CØVER AT LEAST ØNE 0-CØMPØNENT
Step 4
5

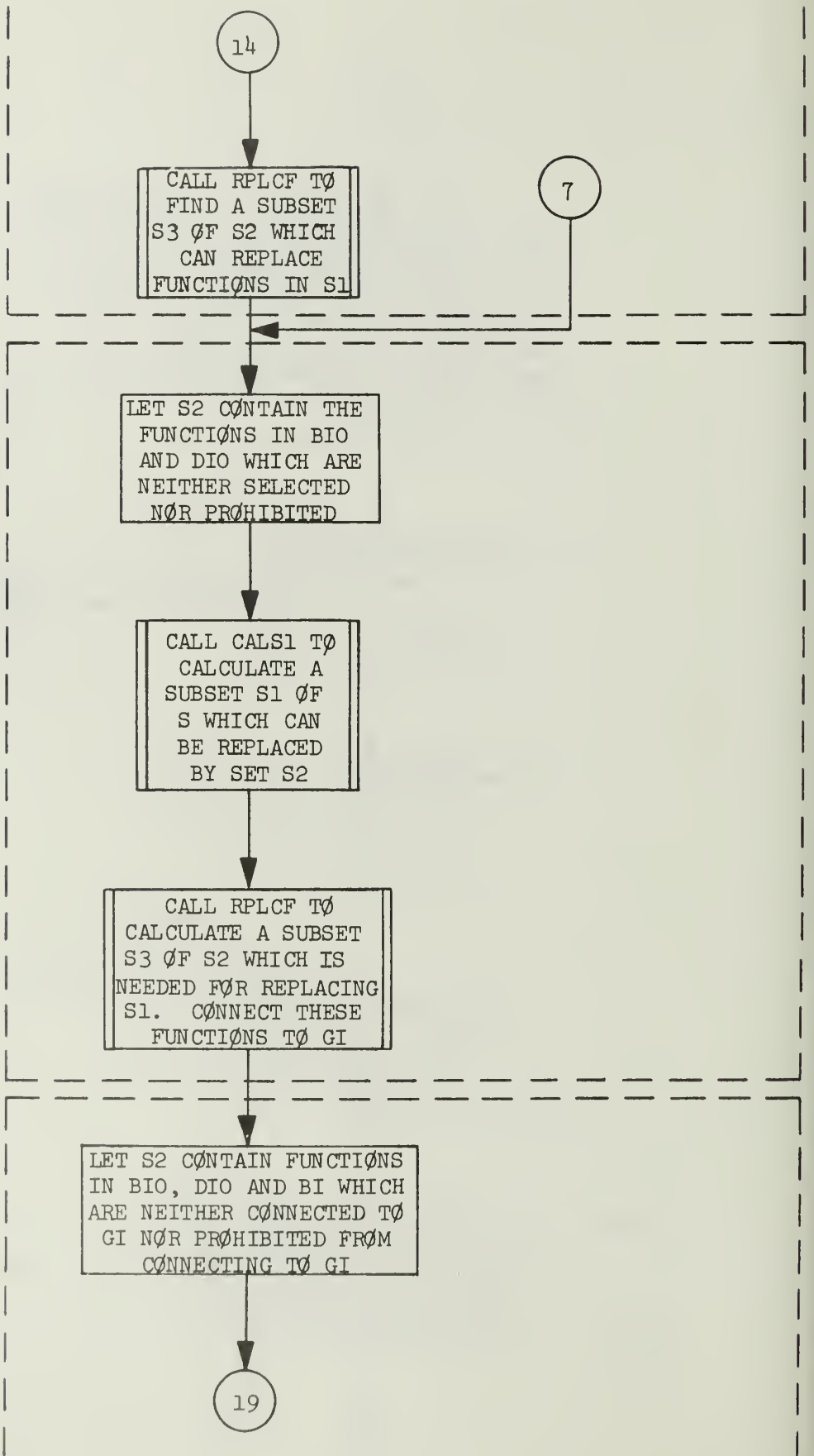

5
ARRANGE BI ACCØRDING TØ THE NUMBER ØF 0-CØMPØNENTS CØVERED
SET S2=BIO ∪ DIO ∪ BI IN THIS ØRDER
LET SET S BE THE EXTERNAL VARIABLES CØNNECTED TØ GI WHICH CØVER AT LEAST ØNE 0-CØMPØNENT IN THE CSPFE ØF GI
S=ϕ
CALL CALS1 TØ CALCULATE A SUBSET S1 ØF S WHICH CAN BE REPLACED BY S2
CALL RPLCF TØ SELECT A SUBSET S3 ØF S2 WHICH IS NECESSARY FØR REPLACING S1
8

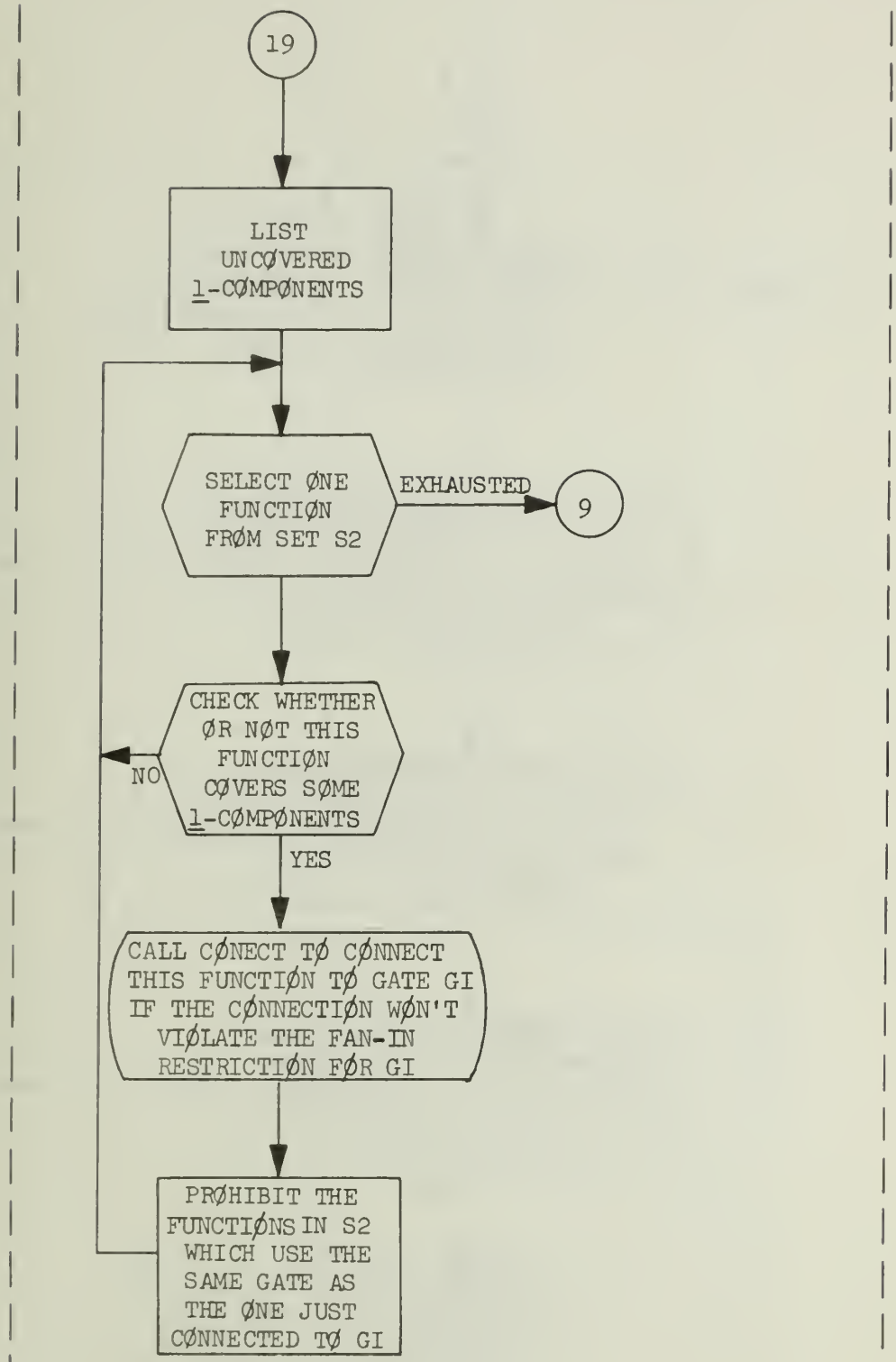
ep 5

Step 6

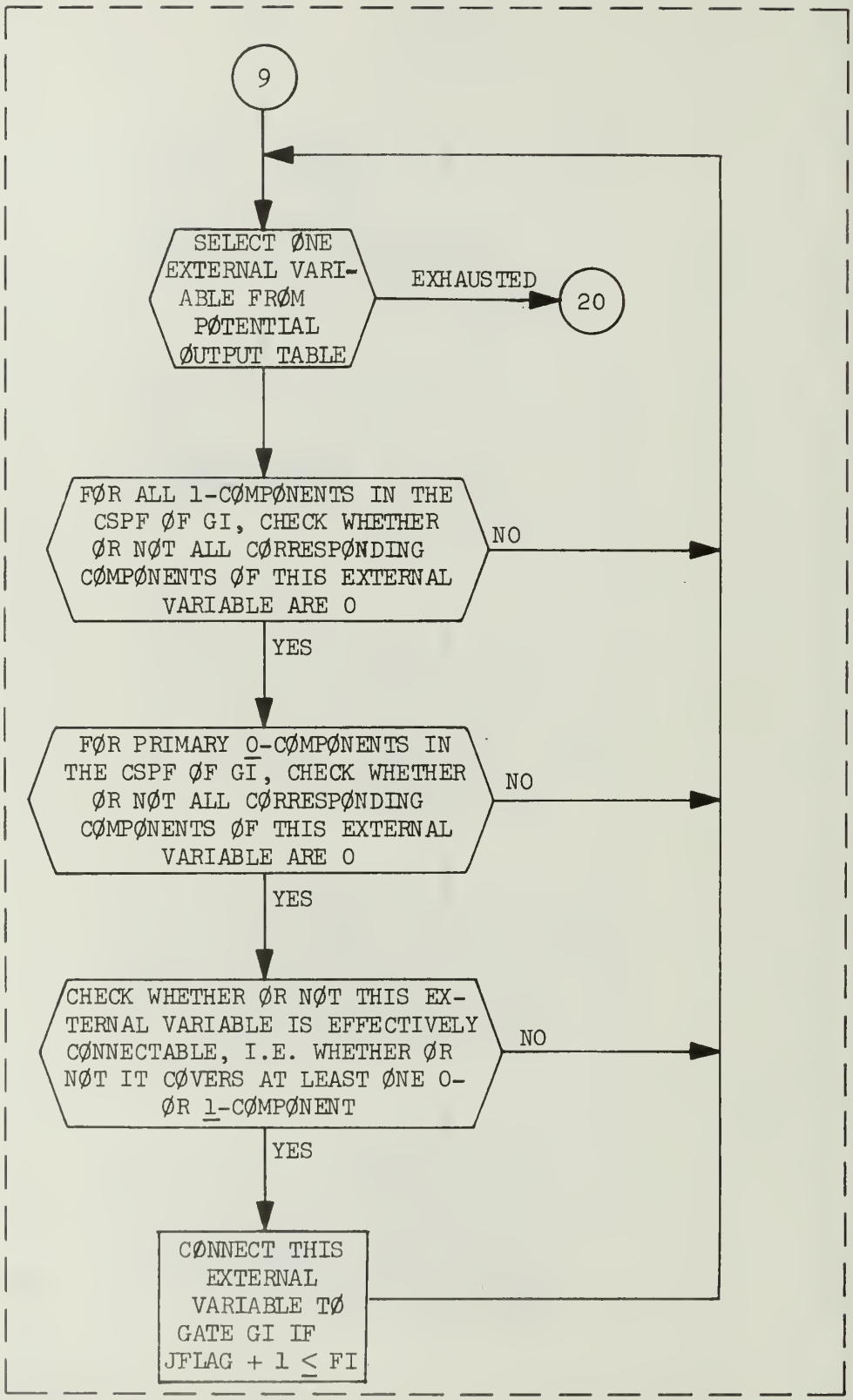

8
CALL ØRDRQ2 TØ CLASSIFY INPUT FUNCTIØNS ØF GI FRØM GATES INTØ TWØ SUBSETS ØNE ØF WHICH CØNTAINS FUNCTIØNS CØVERING AT LEAST ØNE PRIMARY 0-CØMPØNENT, THE ØTHER CØNTAINS ØTHER INPUT FUNCTIØNS CØVERING SØME 0-CØMPØNENTS
LET S CØNTAIN THE FUNCTIØNS IN THE FIRST SUBSET CALCULATED ABØVE
S=ϕ
7
SELECT ØNE FUNCTIØN, GP FRØM SET S
EXHAUSTED
16
LIST ALL PRIMARY 1-CØMPØNENTS IN GP. LET THIS BE SET ES1E.
17
SELECT ØNE CØMPØNENT ES1E(I) FRØM ES1E
EXHAUSTED
18
11

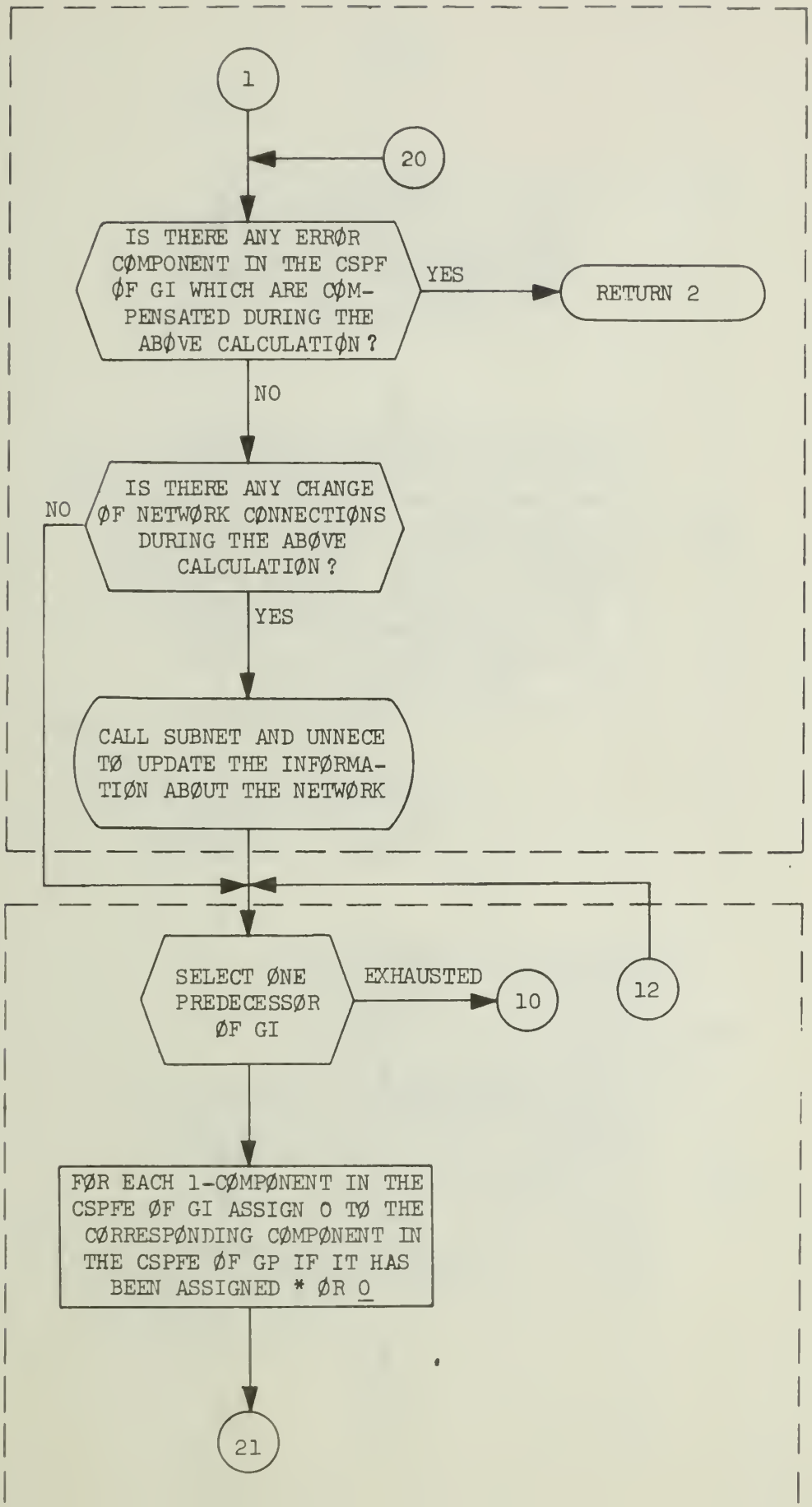

11
SELECT AS A SUBSET S4 ØF S2 THE FUNCTIØNS WHØSE VALUES AT ES1E(I) ARE 0. WHENEVER A FUNCTIØN IS SELECTED, THE FUNCTIØNS IN S2 WHICH USE THE SAME GATE AS THE SELECTED ØNE ARE PRØHIBITED FRØM BEING SELECTED
CHECK WHETHER ØR NØT FUNCTIØNS IN S4 CØVER ALL ESSENTIAL 1'S IN GP
NO
17
RESTØRE S2 RESET S4
YES
PUT GP INTØ S1. ADD ESSENTIAL 1'S CØVERED BY GP INTØ SET T1 SET S2=S4
18
16
T1=ϕ?
NO
YES
ADD INPUT FUNCTIØNS (FROM GATES) ØF GI WHICH CØVER SØME 0-COMPONENTS INTØ SET S
7
CALL CALS1 TØ FIND ADDITIØNAL FUNCTIØNS FØR S1 (SUBSET ØF S) WHICH CAN BE REPLACED BY FUNCTIØNS OF S2
14

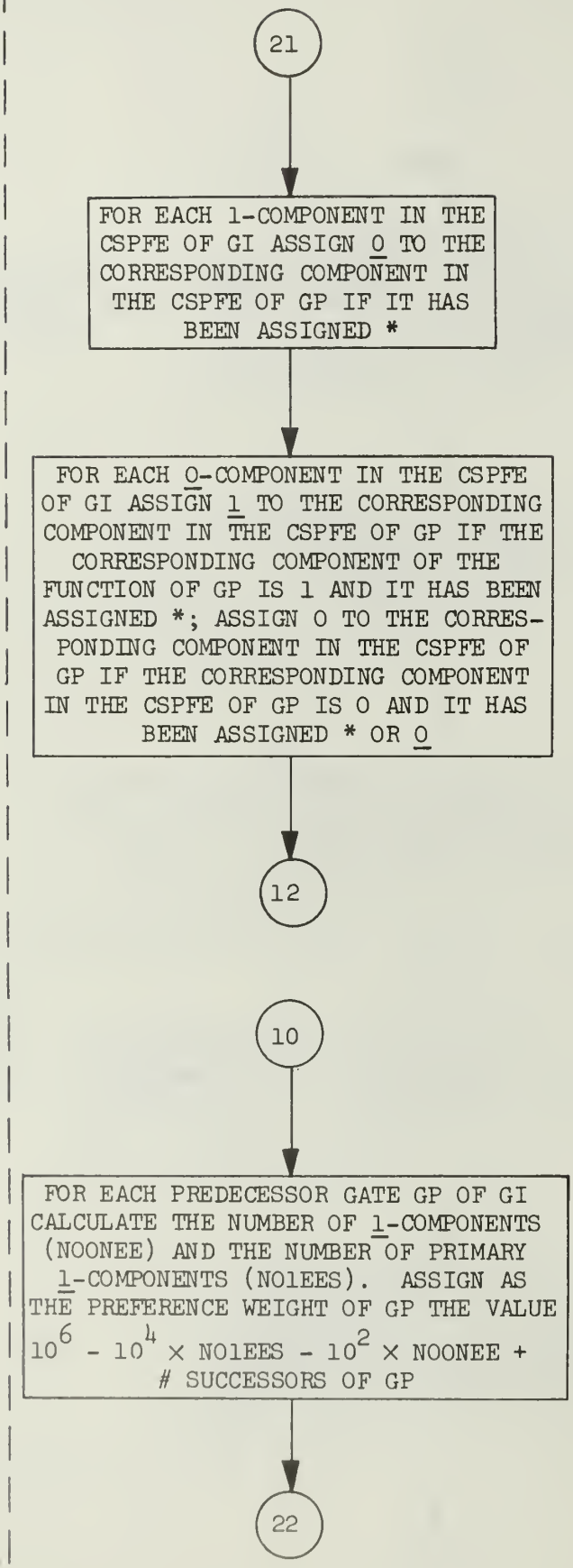

14
CALL RPLCF TØ FIND A SUBSET S3 ØF S2 WHICH CAN REPLACE FUNCTIØNS IN S1
7
LET S2 CØNTAIN THE FUNCTIØNS IN BIO AND DIO WHICH ARE NEITHER SELECTED NØR PRØHIBITED
CALL CALS1 TØ CALCULATE A SUBSET S1 ØF S WHICH CAN BE REPLACED BY SET S2
Step 7
CALL RPLCF TØ CALCULATE A SUBSET S3 ØF S2 WHICH IS NEEDED FØR REPLACING S1. CØNNECT THESE FUNCTIØNS TØ GI
LET S2 CØNTAIN FUNCTIØNS IN BIO, DIO AND BI WHICH ARE NEITHER CØNNECTED TØ GI NØR PRØHIBITED FRØM CØNNECTING TØ GI
Step 8
19

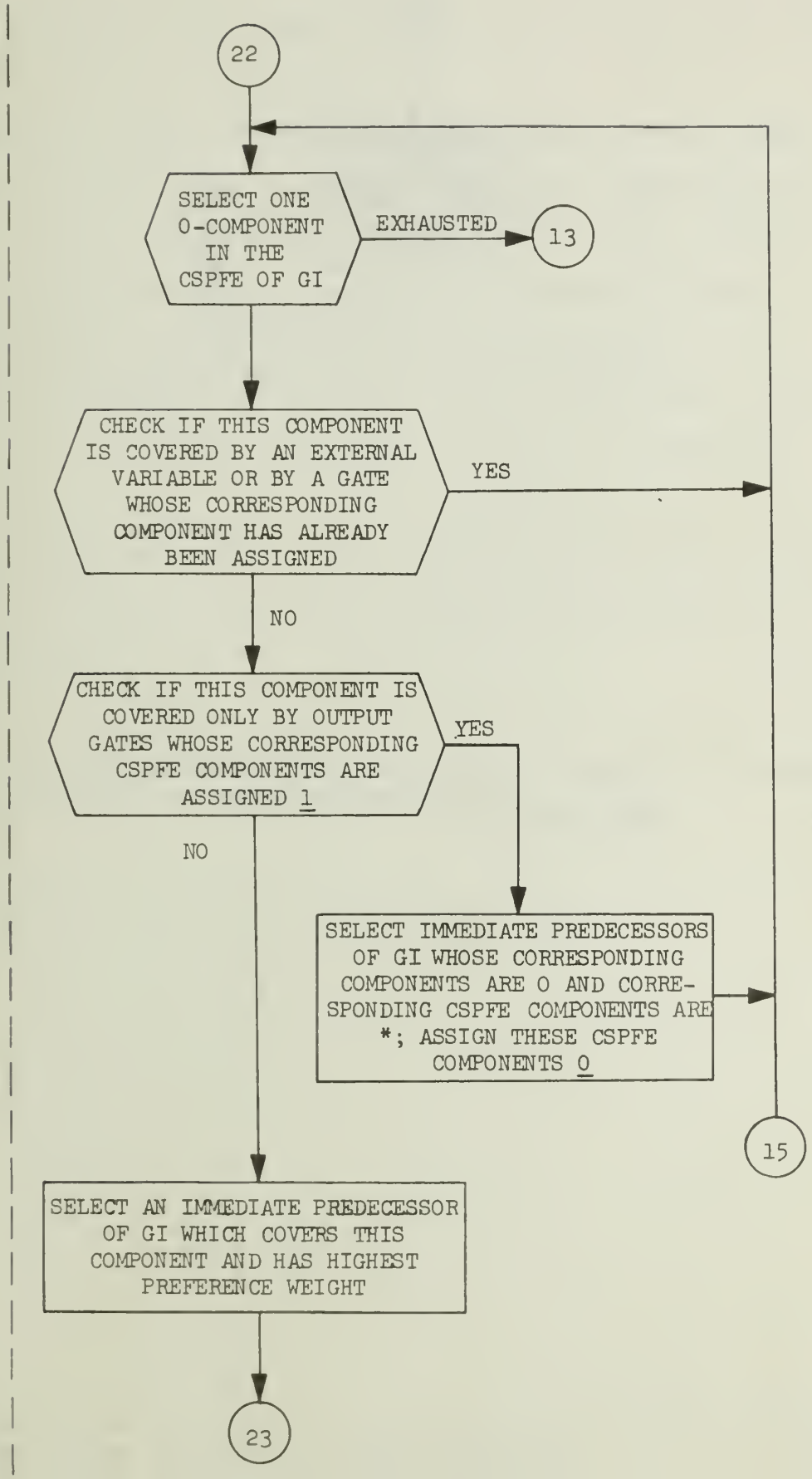

19
LIST UNCØVERED 1-CØMPØNENTS
SELECT ØNE FUNCTIØN FRØM SET S2
EXHAUSTED
9
CHECK WHETHER ØR NØT THIS FUNCTIØN CØVERS SØME 1-CØMPØNENTS
NO
YES
CALL CØNECT TØ CØNNECT THIS FUNCTIØN TØ GATE GI IF THE CØNNECTIØN WØN'T VIØLATE THE FAN-IN RESTRICTIØN FØR GI
PRØHIBIT THE FUNCTIØNS IN S2 WHICH USE THE SAME GATE AS THE ØNE JUST CØNNECTED TØ GI

Step 9


9
SELECT ØNE EXTERNAL VARIABLE FRØM PØTENTIAL ØUTPUT TABLE
EXHAUSTED
20
FØR ALL 1-CØMPØNENTS IN THE CSPF ØF GI, CHECK WHETHER ØR NØT ALL CØRRESPØNDING CØMPØNENTS ØF THIS EXTERNAL VARIABLE ARE 0
NO
YES
FØR PRIMARY 0-CØMPØNENTS IN THE CSPF ØF GI, CHECK WHETHER ØR NØT ALL CØRRESPØNDING CØMPØNENTS ØF THIS EXTERNAL VARIABLE ARE 0
NO
YES
CHECK WHETHER ØR NØT THIS EXTERNAL VARIABLE IS EFFECTIVELY CØNNECTABLE, I.E. WHETHER ØR NØT IT CØVERS AT LEAST ØNE 0- ØR 1-CØMPØNENT
NO
YES
CØNNECT THIS EXTERNAL VARIABLE TØ GATE GI IF JFLAG + 1 ≤ FI


1
20
IS THERE ANY ERRØR CØMPONENT IN THE CSPF ØF GI WHICH ARE CØM-PENSATED DURING THE ABØVE CALCULATIØN ?
YES
RETURN 2
NO
Step 10
IS THERE ANY CHANGE ØF NETWØRK CØNNECTIØNS DURING THE ABØVE CALCULATIØN ?
NO
YES
CALL SUBNET AND UNNECE TØ UPDATE THE INFØRMA-TIØN ABØUT THE NETWØRK
SELECT ØNE PREDECESSØR ØF GI
EXHAUSTED
10
12
Step 11
FØR EACH 1-CØMPØNENT IN THE CSPFE ØF GI ASSIGN 0 TØ THE CØRRESPØNDING CØMPØNENT IN THE CSPFE ØF GP IF IT HAS BEEN ASSIGNED * ØR 0
21


21
FOR EACH 1-COMPONENT IN THE CSPFE OF GI ASSIGN 0 TO THE CORRESPONDING COMPONENT IN THE CSPFE OF GP IF IT HAS BEEN ASSIGNED *
FOR EACH 0-COMPONENT IN THE CSPFE OF GI ASSIGN 1 TO THE CORRESPONDING COMPONENT IN THE CSPFE OF GP IF THE CORRESPONDING COMPONENT OF THE FUNCTION OF GP IS 1 AND IT HAS BEEN ASSIGNED *; ASSIGN 0 TO THE CORRESPONDING COMPONENT IN THE CSPFE OF GP IF THE CORRESPONDING COMPONENT IN THE CSPFE OF GP IS 0 AND IT HAS BEEN ASSIGNED * OR 0
12
10
FOR EACH PREDECESSOR GATE GP OF GI CALCULATE THE NUMBER OF 1-COMPONENTS (NOONEE) AND THE NUMBER OF PRIMARY 1-COMPONENTS (NO1EES). ASSIGN AS THE PREFERENCE WEIGHT OF GP THE VALUE $10^6 - 10^4 \times$ NO1EES $- 10^2 \times$ NOONEE + # SUCCESSORS OF GP
22


22
SELECT ONE 0-COMPONENT IN THE CSPFE OF GI
EXHAUSTED
13
CHECK IF THIS COMPONENT IS COVERED BY AN EXTERNAL VARIABLE OR BY A GATE WHOSE CORRESPONDING COMPONENT HAS ALREADY BEEN ASSIGNED
YES
NO
CHECK IF THIS COMPONENT IS COVERED ONLY BY OUTPUT GATES WHOSE CORRESPONDING CSPFE COMPONENTS ARE ASSIGNED 1
YES
NO
SELECT IMMEDIATE PREDECESSORS OF GI WHOSE CORRESPONDING COMPONENTS ARE 0 AND CORRESPONDING CSPFE COMPONENTS ARE *; ASSIGN THESE CSPFE COMPONENTS 0
15
SELECT AN IMMEDIATE PREDECESSOR OF GI WHICH COVERS THIS COMPONENT AND HAS HIGHEST PREFERENCE WEIGHT
23

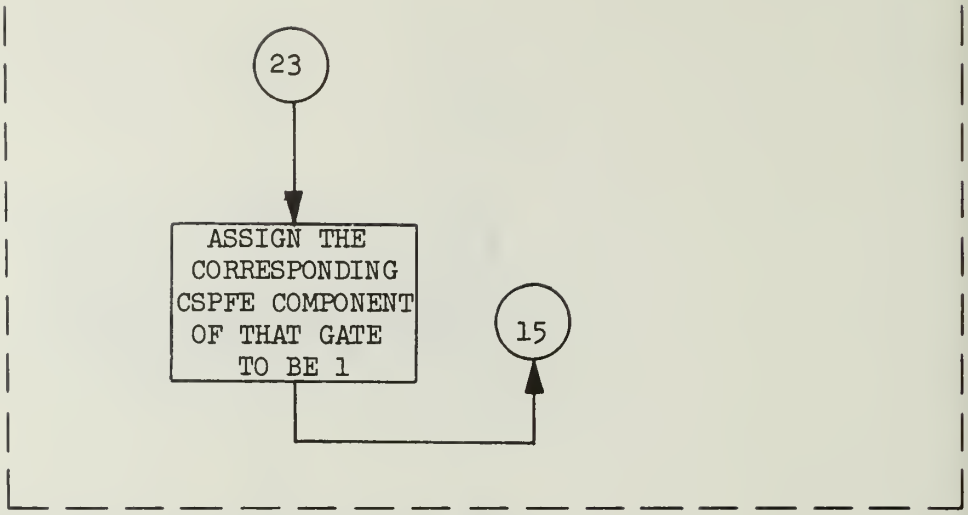


Figure 4.4.5 Flowchart of subroutine RCEC (Dashed blocks are the steps 1, 2, ..., 11 corresponding to those of the error compensation procedures).

Let GK be the gate whose output is going to feed gate GJ to form the potential output.*

If GLEVEL(GI) + DIST + 2 > LEVLIM, then this potential output cannot be used by connecting these gates as shown in Figure 4.4.6, where DIST is the largest number of levels between GK and any predecessor GL of GK. In Figure 4.4.6, the connection of GK to GJ to form the potential output at GJ may increase the level of the network by one and the connection of GJ to GI to compensate for error-components at the output of GI may also increase the level of the network by 1. This is the reason why the check in (2) is different from that in (1).

The modification for the level restriction in subroutine RCEC is shown in the flowchart in Figure 4.4.7, and this flowchart must be included in step 3 in Figure 4.4.5.

*In the potential output table, we store the necessary information to form a potential output but we do not actually make connections.

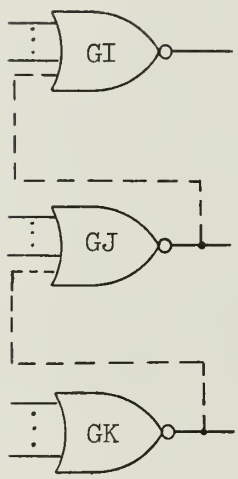


Figure 4.4.6 GK is to be connected to GJ to form the potential output at GJ and GJ is to be connected to GI to compensate for error-components at GI.

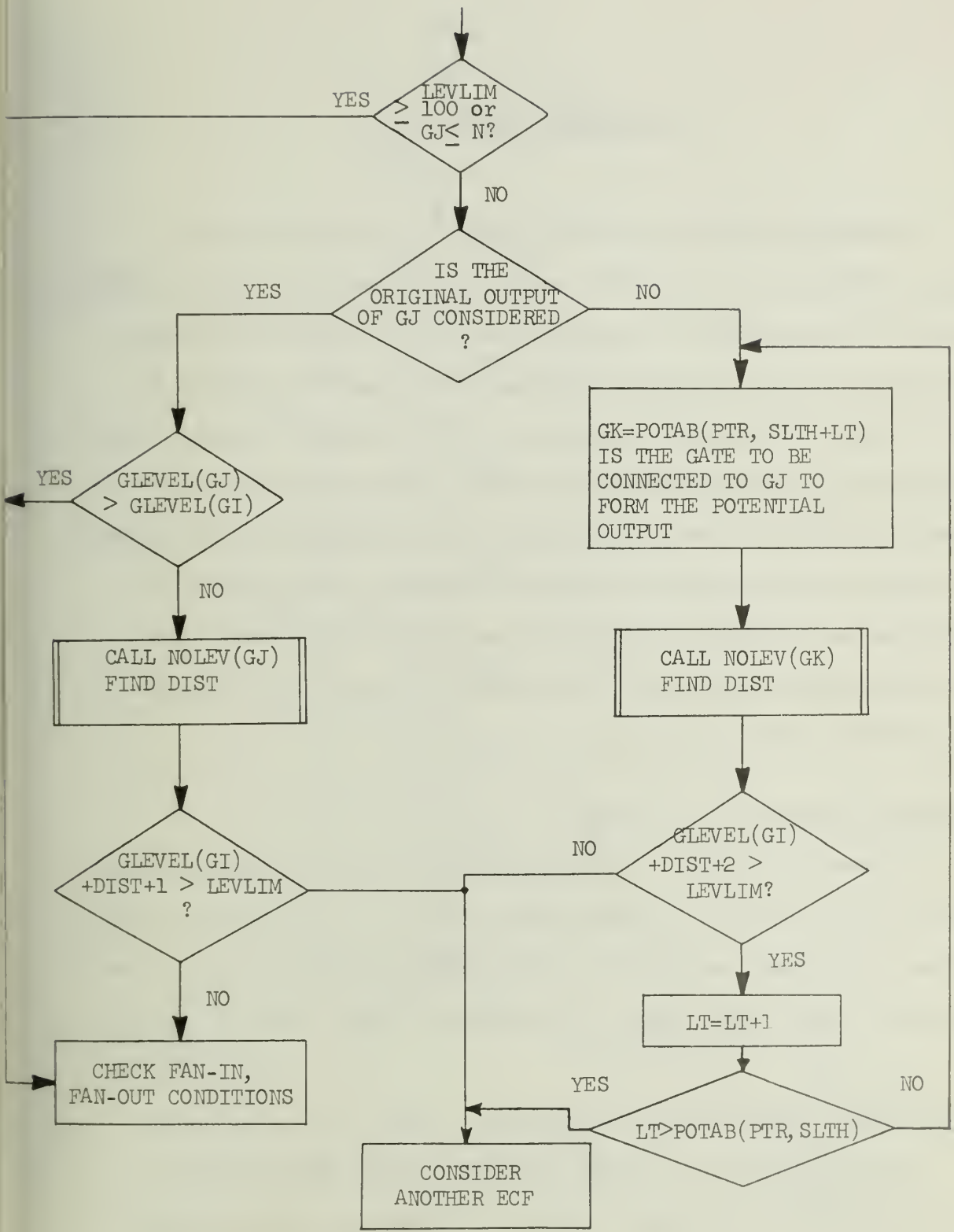


Figure 4.4.7 Modification for level restriction in step 3 of Figure 4.4.5 of subroutine RCEC.

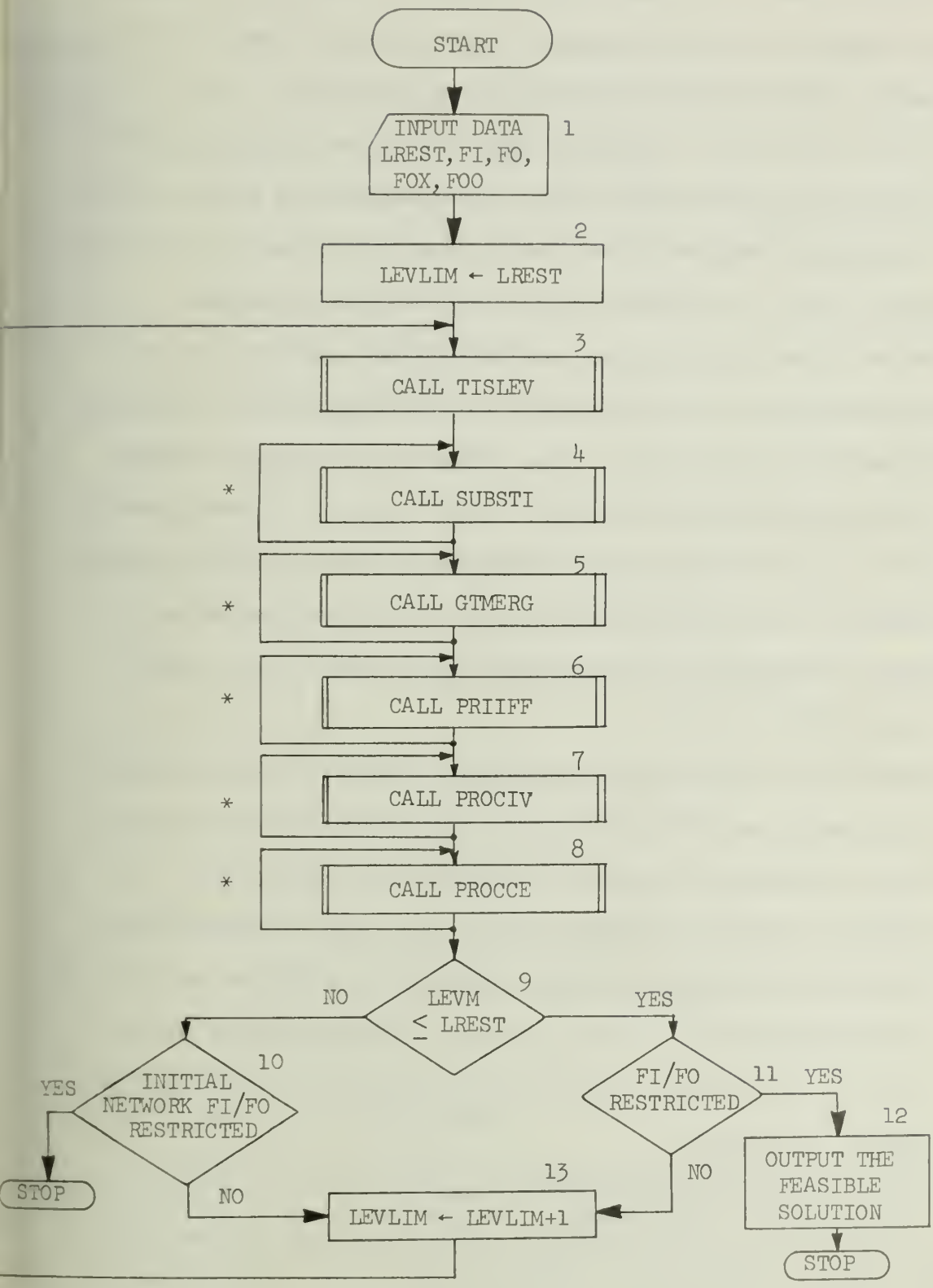
CHAPTER 5
EXPERIMENTAL RESULTS

A FORTRAN program was written to organize the initial network subroutine TISLEV and the modified transduction subroutines. The IBM 360/75J was used to run experiments on the level-restricted transduction program. All subroutines in the transduction program were compiled in FORTRAN H level 21.7 (opt = 2) with the University of Illinois system providing timing subroutines with entry names STIMEZ and KTIMEZ. The results of these experiments are given in section 5.2. Section 5.1 outlines the organization of the control subroutine MAIN. Section 5.3 provides the comparison of effectiveness and efficiency between the level-restricted transduction program (presented in this paper) and the logic design program based on branch-and-bound method [18,19].

5.1 Outline of the Computer Program

The general flowchart of subroutine MAIN is shown in Figure 5.1.1. It outlines the order that subroutine MAIN calls all other subroutines and it also shows the approach used to try to find feasible networks (i.e., networks which satisfy given restrictions). The detail of Figure 5.1.1 is explained below:

Block 1: Read in functions and restrictions. LREST is the given level restriction for the current problem. FI, FO, FOX and FOO are maximum fan-in, maximum fan-out for gates (not output gates), maximum fan-out for external variables and maximum fan-out for output gates, respectively.



* Iterate until there is no further improvment in cost.

Figure 5.1.1 General flowchart of the control subroutine MAIN.

Block 2: LEVLIM is set to the same value as LREST. LEVLIM will temporarily be used as a level limit during the processes of finding the initial network and applying the transduction procedures.

Block 3: The initial network subroutine TISLEV is called to obtain a level-restricted initial network (the number of levels is at most LEVLIM). This initial network may not satisfy the fan-in/fan-out restrictions.

Block 4 through block 8: The transduction subroutines based on gate substitution (SUBSTI), gate merging (GTMERG), connectable and disconnectable functions (PRIIFF and PROCIV) and error-compensation (PROCCE) are called one by one. The primary difference between PRIIFF and PROCIV is that PROCIV concentrates on removing specific gates from the network under consideration whereas PRIIFF does not attempt to remove specific gates [6,8]. The loops marked with a single asterisk (*) will be repeated until there is no further improvement in cost.

Block 9: The network obtained at this point is checked to see whether it is level-restricted (LEVM is the number of levels the network has). If it is, then go to block 11; otherwise go to block 10.

Block 10: Since the network obtained at this point is not level-restricted, check the corresponding initial network to see whether it is fan-in/fan-out restricted. If it is, then no feasible network can be found by this program. Otherwise, go to block 13 to consider the relaxation problem.

Block 11: Check whether the fan-in/fan-out restrictions are satisfied or not. If this is true, then a feasible network has been obtained, and go to block 12. Otherwise, go to block 13.

Block 12: Print out the feasible network found in block 11 and then stop.

Block 13: When this block is reached, the network obtained at this point must belong to one of the following two cases:

(1) It is level-restricted but not fan-in/fan-out restricted.

(2) It is not level-restricted and its corresponding initial network is not fan-in/fan-out restricted.

For each of these two cases, the relaxation problem is considered, i.e., increase LEVLIM by 1 and go back to block 3 to find another initial network with a higher level limit. This initial network, apparently, will not be level-restricted, but it may have less fan-in/fan-out problems than those initial networks with a lower level limit. Starting from this initial network, the number of levels of the network may be reduced after applying the transduction procedures.

It was stated in Chapter 2 that there may not exist any network which satisfies both the level restriction and the fan-in/fan-out restrictions. The organization of subroutine MAIN in Figure 5.1.1 does not guarantee that a feasible network can be found even if there does exist optimal networks. But the statistics in the next section shows that the results derived by this approach are reasonably good.

5.2 Results of the Computer Experiments

The functions used in Chapter 3 were run for experiments. Table 5.2.1 through Table 5.2.11 provide the results. In each table, the function entitled "FUNCTION(HEX)" gives the hexadecimal representation of the truth table of each function. The cost is defined as 1000 x R + C where R is the number of gates and C is the number of connections. The feasibility and

Table 5.2.1 10 Four-variable Functions

RESTRICTION / COST & TIME / FUNCTION (HEX)	FI = FO = FOX = FOO = 2; uncomplemented inputs only		
	LEVLIM = 5	LEVLIM = 6	LEVLIM = 7
	INITIAL → FINAL TIME	INITIAL → FINAL TIME	INITIAL → FINAL TIME
1. 4AF1	17028(NF) → 14025(NF) 355 CS	19030(F) → 17027(F) 533 CS	
2. FBFE	10016(NF) → 10016(NF) 151 CS	12018(F) → 9016(F) 185 CS	
3. ABCE	15025(NF) → 12022(F) 395 CS		
4. 2D8B	22038(NF) → 21037(NF) 673 CS	23037(NF) → 20033(F) 707 CS	24038(F) → 22035(F) 965 CS
5. 9DA5	17029(NF) → 17028(F) 423 CS		
6. 5F12	9014(F) → 9014(F) 113 CS		
7. F1F4	9015(F) → 9015(F) 117 CS		
8. 6830	16026(NF) → 14023(F) 348 CS		
9. 9048	17029(NF) → 16027(NF) 437 CS	23036(NF) → 20032(NF) 1173 CS	24037(F) → 18028(F) 893 CS
10. EA9B	15025(NF) → 14024(NF) 349 CS	18028(NF) → 18028(NF) 508 CS	20030(F) → 17027(F) 847 CS

Table 5.2.2 10 Four-variable Functions

RESTRICTION / COST & TIME / FUNCTION (HEX)	FI = FO = FOX = FOO = 3; uncomplemented inputs only	
	LEVLIM = 4	LEVLIM = 5
	INITIAL → FINAL TIME	INITIAL → FINAL TIME
1. 4AF1	10020(F) → 8021(F) 179 CS	
2. FBFE	8014(F) → 8014(F) 113 CS	
3. ABCE	11022(NF) → 9020(NF) 270 CS	11021(F) → 7017(F) 271 CS
4. 2D8B	15031(NF) → 14029(NF) 553 CS	16030(F) → 13026(F) 428 CS
5. 9DA5	9018(F) → 7018(F) 127 CS	
6. 5F12	7013(F) → 7013(F) 104 CS	
7. F1F4	7014(F) → 7014(F) 113 CS	
8. 6830	12023(F) → 11021(F) 314 CS	
9. 9048	14027(NF) → 13025(NF) 340 CS	14026(F) → 14026(F) 433 CS
10. EA9B	10020(F) → 9019(F) 188 CS	

Table 5.2.3 10 Five-Variable Functions

RESTRICTION / COST & TIME / FUNCTION (HEX)	FI = FO = FOX = FOO = 4; complemented and uncomplemented inputs	
	LEVLIM = 3	LEVLIM = 4
	INITIAL → FINAL TIME	INITIAL → FINAL TIME
1. 4AF295F6	14041(NF) → 11036(NF) 641 CS	14039(F) → 11033(F) 604 CS
2. A6CDDF18	17048(NF) → 11036(NF) 688 CS	16044(F) → 12036(F) 861 CS
3. FF68A153	18048(NF) → 12037(NF) 584 CS	19046(F) → 11032(F) 1271 CS
4. 1EE65240	16041(NF) → 12032(F) 539 CS	
5. 9E63BE75	13038(NF) → 10033(NF) 391 CS	18043(F) → 10031(F) 843 CS
6. 0A888103	11027(F) → 8026(F) 274 CS	
7. 49F363CD	14041(NF) → 10028(NF) 372 CS	14039(F) → 9027(F) 756 CS
8. 8B5809F0	11033(F) → 10030(F) 465 CS	
9. BFD6C6DA	20053(NF) → 12036(NF) 593 CS	19049(F) → 13035(F) 1333 CS
10. C6E7103E	14037(NF) → 11031(F) 568 CS	

Table 5.2.4 10 Five-variable Functions

RESTRICTION / COST & FUNCTION (HEX)	FI = FO = FOX = FOO = 3; complemented and uncomplemented inputs	
	LEVLIM = 4	LEVLIM = 5
	INITIAL → FINAL TIME	INITIAL → FINAL TIME
1. 4FA295F6	19043(F) → 19043(F) 952 CS	
2. A6CDDF18	18042(F) → 16040(F) 1004 CS	
3. FF68A153	18042(F) → 13033(F) 1263 CS	
4. 1EE65240	18041(NF) → 12030(F) 572 CS	
5. 9E63BE75	19040(F) → 16036(F) 1161 CS	
6. 0A888103	12028(F) → 11026(F) 454 CS	
7. 49F363CD	21047(NF) → 14033(F) 1553 CS	
8. 8B5809F0	16036(F) → 11028(F) 738 CS	
9. BFD6C6DA	30065(NF) → 21048(NF) 2178 CS	33068(F) → 14036(F)* 2195 CS
10. C6E7103E	22046(F) → 13034(F) 1205 CS	

*The number of levels of this network is reduced to 4.

Table 5.2.5 One-bit Full Adder

RESTRICTIONS / COST & TIME / FUNCTIONS (HEX)	FI = FO = FOX = FOO = 3; uncomplemented inputs only
	LEVLIM = 4
	INITIAL COST → FINAL COST TIME
1. 69	15029(F) → 12024(F)
2. 17	362 CS

Table 5.2.6 One-bit Full Adder

RESTRICTIONS / COST & TIME / FUNCTIONS (HEX)	FI = FO = FOX = FOO = 3; complemented and uncomplemented inputs
	LEVLIM = 3
	INITIAL COST → FINAL COST TIME
1. 69	8020(F) → 8020(F)
2. 17	140 CS

Table 5.2.7 Two-bit Multiplier

RESTRICTIONS / COST & TIME / FUNCTIONS (HEX)	FI = FO = FOX = FOO = 3; uncomplemented inputs only	
	LEVLIM = 4	LEVLIM = 5
	INITIAL → FINAL TIME	INITIAL → FINAL TIME
1. 0505 2. 0356 3. 0032 4. 0001	13026(NF) → 13026(NF) 372 CS	11021(F) → 11021(F) 292 CS

Table 5.2.8 Two-bit Multiplier

RESTRICTIONS / COST & TIME / FUNCTIONS (HEX)	FI = FO = FOX = FOO = 3; complemented and uncomplemented inputs	
	LEVLIM = 3	LEVLIM = 4
	INITIAL → FINAL TIME	INITIAL → FINAL TIME
1. 0505 2. 0356 3. 0032 4. 0001	9022(NF) → 9022(NF) 224 CS	7017(F) → 7017(F) 164 CS

Table 5.2.9 Two-bit Full Adder

RESTRICTIONS / COST & TIME / FUNCTIONS (HEX)	FI = FO = FOX = FOO = 4; uncomplemented inputs only	
	LEVLIM = 4	LEVLIM = 5
	INITIAL → FINAL TIME	INITIAL → FINAL TIME
1. 66996699	38089(NF) → 33079(NF)	24053(F) → 21049(F)
2. 1E78E187		
3. 01071F7F	7978 CS	2891 CS

Table 5.2.10 Two-bit Full Adder

RESTRICTIONS / COST & TIME / FUNCTIONS (HEX)	FI = FO = FOX = FOO = 4; complemented and uncomplemented inputs	
	LEVLIM = 3	LEVLIM = 4
	INITIAL → FINAL TIME	INITIAL → FINAL TIME
1. 66996699	33086(NF) → 29079(NF)	24064(F) → 14041(F)
2. 1E78E187		
3. 01071F7F	4195 CS	3198 CS

Table 5.2.11 Su-Nam's Example

RESTRICTIONS / COST & TIME / FUNCTIONS (HEX)	FI = FO = FOX = FOO = 2; complemented and uncomplemented inputs		
	LEVLIM = 4	LEVLIM = 5	LEVLIM = 6
	INITIAL → FINAL TIME	INITIAL → FINAL TIME	INITIAL → FINAL TIME
4 functions which contain don't cares	22040(NF) → 19046(NF) 1157 CS	20037(NF) → 20036(NF) 767 CS	21037(F) → 21037(F) 977 CS

4 functions used:

1. 000* 1*11 01*1 1111
2. 011* 1*11 0011 0000
3. 001* 1*00 001* *000
4. 001* 1*01 0111 1111

infeasibility of a network is indicated by (F) and (NF), respectively, following the cost. A feasible network is a network which satisfies all given restrictions. In each table, the maximum fan-in/fan-out and the availability of complemented or uncomplemented external variables are shown in the top row.

According to section 5.1, an initial network will be obtained under the given level restriction (LREST). Five transduction procedures will, then, be employed to simplify this initial network. If the simplified network is not feasible, then the relaxation problem will be considered. The costs of level-restricted initial networks, the costs of networks after applying transduction procedures and computation time are listed in the first column entitled " LEVLIM = X " INITIAL → FINAL . The results for the relaxation problems are shown TIME in the following columns in each table. An example is Function #1 in Table 5.2.1, where under LEVLIM = 5 and FI = FO = FOX = FOO = 2 an infeasible initial network with 17 gates and 28 connections is produced. After applying the five transduction procedures, the cost is reduced to 14 gates and 25 connections, but it is still infeasible. Therefore the relaxation problem with LEVLIM = 6 is considered, and a feasible initial network with 19 gates and 30 connections is obtained. After the application of the transduction procedures, the cost is reduced to 15 gates and 25 connections. This is a feasible network with respect to LEVLIM = 6, but not a feasible network with respect to LEVLIM = 5.

Sometimes, during the consideration of the relaxation problems, the number of levels of initial networks may be reduced and a feasible network (with respect to the given limit (LREST) can be obtained. A typical example is Function #9 in Table 5.2.4. In this problem, when LEVLIM = 4, no feasible

solution can be found. But when the relaxation problem with LEVLIM = 5 is considered, the number of levels of the network is reduced to 4 after applying the transduction procedures. Hence this network (with 14 gates and 36 connections) becomes feasible with respect to LEVLIM = 4.

Table 5.2.1 gives the results of 10 four-variable functions with FI = FO = FOX = FOO = 2. When LEVLIM = 5, there are 5 feasible solutions (Functions #3, #5, #6, #7 and #8). It is noticed that initial networks of problems #3, #5 and #8 are infeasible; but after the application of the transduction procedures, some fan-in/fan-out problems are solved and networks become feasible. When LEVLIM = 6, there are 8 feasible solutions. When LEVLIM = 7, there are ten. Table 5.2.2 gives the results of the same four-variable functions as those in Table 5.2.1 but with different fan-in/fan-out restrictions. When LEVLIM = 4, seven feasible solutions are found; when LEVLIM = 5, ten feasible solutions are found.

Table 5.2.3 and Table 5.2.4 provide the results of ten five-variable functions with different fan-in/fan-out and level restrictions. It is observed from Table 5.2.1 through Table 5.2.4 that if the maximum fan-in and fan-out are higher, then more feasible networks can usually be obtained for the same level restriction. This is also true if the level limit is higher but the fan-in/fan-out restrictions are the same. The computation time has a similar tendency--if the level limit is the same but the fan-in/fan-out are higher, then less computation time is required.

Table 5.3.5 through Table 5.3.11 give the results for four multiple-output functions: one-bit full adder, two-bit full adder, two-bit multiplier and Su-Nam's example [23]. The effects of the fan-in/fan-out restrictions and the level restriction on the number of feasible networks and the computation time are similar to that of the single-output functions.

The Su-Nam's network has 25 gates, 42 connections and 6 levels. The level-restricted transduction program can get a network with the same number of levels but with 4 gates and 5 connections save in cost.

5.3 Comparisons and Conclusions

Network transduction procedures are designed for the purpose of obtaining near-optimal networks in reasonable computation time. In order to find out the effectiveness and the efficiency of the level-restricted transduction program the results obtained in Table 5.2.2 and Table 5.2.4 are compared with the results obtained by the logical design program based on the branch-and-bound method which can either produce optimal networks or prove the infeasibility. Table 5.3.1 and Table 5.3.2 provide the comparisons. In Table 5.3.1, the transduction program obtained feasible networks for seven functions. The logical design program based on the branch-and-bound method obtained feasible networks for nine functions in two minutes. Among these nine feasible solutions, seven are really optimal. For function #4, no feasible network was obtained and for function #8 and function #9, only intermediate results were obtained after two minutes execution. For functions #5, #6 and #7, the transduction program derived optimal networks but needed less computation time; for all other functions, the transduction program gave worse results. These observations show that the logical design program based on the branch-and-bound method are more effective than the transduction program in finding optimal networks for four-variable single-output problems.

In Table 5.3.2, however, the transduction program obtained feasible networks for all functions. But the logical design program based on the branch-and-bound method could not obtain any optimal network after two minutes execution; it only produced two intermediate networks. For function #7, the

Table 5.3.1 10 Four-variable Functions

RESTRICTIONS / COST & TIME / FUNCTION (HEX)	FI = FO = FOX = FOO = 3; uncomplemented inputs only LEVLIM = 4	
	Results by The Transduction Programs	Optimal Solutions by The Branch-and-Bound Method
1. 4AF1	8022 179 CS	8019 704 CS
2. FBFE	8014 113 CS	6012 48 CS
3. ABCE	NO FEASIBLE SOLUTION	8018 2436 CS
4. 2D8B	NO FEASIBLE SOLUTION	NO RESULTS AFTER TWO MINUTES
5. 9DA5	7018 127 CS	7018 278 CS
6. 5F12	7013 104 CS	7013 709 CS
7. F1F4	7014 113 CS	7014 133 CS
8. 6830	11021 314 CS	INTERMEDIATE RESULTS 8017 12000 CS
9. 9048	NO FEASIBLE SOLUTION	INTERMEDIATE RESULTS 9021 12000 CS
10. EA9B	9019 188 CS	9018 3359 CS

Table 5.3.2 10 Five-variable Functions

RESTRICTIONS / COST & TIME / FUNCTION (HEX)	FI = FO = FOX = FOO = 3; LEVLIM = 4; complemented and uncomplemented inputs	
	RESULTS BY THE TRANSDUCTION PROGRAM	RESULTS BY THE BRANCH-AND-BOUND METHOD
1. 4FA295F6	19043 952 CS	
2. A6CDDF18	16040 1004 CS	NO RESULTS
3. FF68A153	13033 1263 CS	AFTER TWO MINUTES
4. 1EE65240	12030 572 CS	
5. 9E63BE75	16036 1161 CS	INTERMEDIATE RESULTS 14038 12000 CS
6. 0A888103	11026 454 CS	NO RESULTS AFTER TWO MINUTES
7. 49F363CD	14033 1553 CS	INTERMEDIATE RESULTS 15039 12000 CS
8. 8B5809F0	11028 738 CS	
9. BFD6C6DA	14036 4373 CS	NO RESULTS AFTER TWO MINUTES
10. C6E7103E	13034 1205 CS	

etwork obtained by the transduction program is even better than that btained by the logical design program based on the branch-and-bound method. herefore, the transduction program is much more efficient and effective in he case of large networks (more than 9 gates).

Besides the branch-and-bound method, the logical design based on the mplicit enumeration method [17] can also obtain optimal networks and prove he infeasibility. But the implicit enumeration method is generally more ime-consuming than the branch-and-bound method, though it has flexibility n changing gate types and is easier to program. The map factoring method [16] an take into account both the level restriction and the fan-in/fan-out estrictions, but this method is convenient to use only when the network is mall. No other existing algorithms consider both the level restriction and he fan-in/fan-out restrictions at the same time.

LIST OF REFERENCES

[1] Culliney, J. N., H. C. Lai, and Y. Kambayashi, "Pruning Procedures for NOR Networks using Permissible Functions (Principles of NOR Networks Transduction Programs NETTRA-PG1, NETTRA-P1, and NETTRA-P2)," Report No. UIUCDCS-R-74-690, Department of Computer Science, University of Illinois, Urbana, Illinois, November, 1974.

[2] Kambayashi, Y. and J. N. Culliney, "NOR Network Transduction Procedures Based on Connectable and Disconnectable Conditions (Principles of NOR Network Transduction Programs NETTRA-G1 and NETTRA-G2)," to appear as a report, Department of Computer Science, University of Illinois.

[3] Cutler, R. B., et al., "TISON-VERSION 1: A Program to Derive a Minimal Sum of Products for a Function or Set of Functions which may not be Completely Specified," to appear as a report, Department of Computer Science, University of Illinois, Urbana, Illinois.

[4] Gimpel, James F., "The Minimization of TANT Networks," IEEE Transactions on Electronic Computers, Vol. EC-16, pp. 18-38, February, 1967.

[5] Hohulin, K. R. and S. Muroga, "Alternative Methods for Solving the CC-table in Gimpel's Algorithm for Synthesizing Optimal Three Level NAND Networks," Report No. UIUCDCS-R-75-720, Department of Computer Science, University of Illinois, Urbana, Illinois, April, 1975.

[6] Hohulin, K. R., "Network Transduction Programs Based on Connectable and Disconnectable Conditions with Fan-in and Fan-out Restrictions (a Description of NETTRA-G1-FIFO and NETTRA-G2-FIFO)," Report No. UIUCDCS-R-75-719, Department of Computer Science, University of Illinois, Urbana, Illinois, April, 1975.

[7] Hu, K. C., "NOR (NAND) Network Design: Error-compensation Procedures for Fan-in and Fan-out Restricted Networks (NETTRA-E1-FIFO and NETTRA-E2-FIFO)," to appear as Report, Department of Computer Science, University of Illinois, Urbana, Illinois.

[8] Culliney, J. N., "Program Manual: NOR Network Transduction Based on Connectable and Disconnectable Conditions (Reference Manual of NOR Network Transduction Programs NETTRA-G1 and NETTRA-G2)," Report No. UIUCDCS-R-74-698, Department of Computer Science, University of Illinois, February, 1975.

[9] Kambayashi, Y. and S. Muroga, "Network Transduction by Permissible Functions (General Principles of NOR Network Transduction NETTRA Programs)," Report No. UIUCDCS-R-76-804, Department of Computer Science, University of Illinois, Urbana, Illinois, June, 1976.

[10] Kambayashi, Y., H. C. Lai, J. N. Culliney and S. Muroga, "NOR Network Transduction by Error-compensation (Principles of NOR Network Transduction Programs NETTRA-E1, NETTRA-E2, and NETTRA-E3)," Report No. UIUCDCS-R-75-737, Department of Computer Science, University of Illinois, Urbana, Illinois.

[11] Lai, H. C. and Y. Kambayashi, "Generalized Gate Merging and Substitution for NOR Network Transduction (Principles of NOR Network Transduction Programs NETTRA-G3 and NETTRA-G4)," to appear as Report, Department of Computer Science, University of Illinois, Urbana, Illinois.

[12] Lai, H. C. and J. N. Culliney, "Program Manual: NOR Network Transduction Based on Error-compensation (Reference Manual of NOR Network Transduction Programs NETTRA-E1, NETTRA-E2 and NETTRA-E3)," Report No. UIUCDCS-R-75-732, Department of Computer Science, University of Illinois, Urbana, Illinois, June, 1975.

[13] Lai, H. C., "Program Manual: NOR Network Transduction by Generalized Gate Merging and Substitution (Reference Manual of NOR Network Transduction Programs NETTRA-G3 and NETTRA-G4)," to appear as Report, Department of Computer Science, University of Illinois, Urbana, Illinois.

[14] Lai, H. C. and J. N. Culliney, "Program Manual: NOR Network Pruning Procedures using Permissible Functions (Reference Manual of NOR Network Transduction Programs NETTRA-PG1, NETTRA-P1, and NETTRA-P2)," Report No. UIUCDCS-R-74-686, Department of Computer Science, University of Illinois, November, 1974.

[15] Legge, J. G., "The Design of NOR Networks under Fan-in and Fan-out Constraints (A Program Manual for FIFOTRAN-G1)," Report No. 661, Department of Computer Science, University of Illinois, Urbana, Illinois.

[16] Maley, G. A. and J. Earle, The Logical Design of Transistor Digital Computers. The NOR network constructed in the same manner as the NAND tree network in Fig. 6.4.1, page 156, is called the universal network in the text. Prentice Hall, 1963.

[17] Muroga, S. and T. Ibaraki, "Design of Optimal Switching Networks by Integer Programming," IEEE Trans. Comput., Vol. C-21, pp. 573-582, June, 1972.

[18] Nakagawa, T. and H. C. Lai, "A Branch-and-bound Algorithm for Optimal NOR Networks (The Algorithm Description)," Report No. UIUCDCS-R-71-438, Department of Computer Science, University of Illinois, Urbana, Illinois, April, 1971.

[19] Nakagawa, T. and H. C. Lai, "Reference Manual of FORTRAN Program ILLOD-(NOR-B) for Optimal NOR Networks," Report No. UIUCDCS-R-71-488, Department of Computer Science, University of Illinois, Urbana, Illinois, December, 1971.

[20] Nakagawa, T. and S. Muroga, "Exposition of Davidson's Thesis 'An Algorithm for NAND Decomposition of Combinational Switching Systems'," File UIUCDCS-F-71-869, Department of Computer Science, University of Illinois, Urbana, Illinois, August, 1969.

[21] Nakagawa, T. and S. Muroga, "Comparison of the Implicit Enumeration Method and the Branch-and-bound Method for Logical Design," Report No. UIUCDCS-R-71-455, Department of Computer Science, University of Illinois, Urbana, Illinois, June, 1971.

[22] Plangsiri, B., "NOR Network Transduction Procedures: Merging of Gates and Substitution of Gates for Fan-in and Fan-out Restricted Networks (NETTRA-G3-FIFO and NETTRA-PG1-FIFO)," Report No. UIUCDCS-R-74-688, Department of Computer Science, University of Illinois, Urbana, Illinois, December, 1974.

[23] Su, Y. H. and C. W. Nam, "Computer-aided Synthesis of Multiple Output Multi-level NAND Networks with Fan-in and Fan-out Constraints," IEEE Trans. Comput., Vol. C-20, December, 1971.

GRAPHIC DATA	1. Report No. UIUCDCS-R-77-849	2.	3. Recipient's Accession No.
and Subtitle			5. Report Date January, 1977
Level-Restricted NOR Network Transduction Procedures			6.
(s) C. C. Hu			8. Performing Organization Rept. No.
ming Organization Name and Address			10. Project/Task/Work Unit No.
partment of Computer Science iversity of Illinois ana, Illinois			11. Contract/Grant No. NSF DCR73-03421 A01
nsoring Organization Name and Address			13. Type of Report & Period Covered
tional Science Foundation shington, DC			14.

lementary Notes

racts

The NOR network transduction procedures can be grouped into three classes, ding to their characteristics: pruning procedures, general procedures, and error-nsation procedures. These procedures are implemented into ten programs: A-PG1, -P1, -P2, -G1, -G2, -G3, -G4, -E1, -E2, and -E3. Among these ten programs, -PG1, -G1, -G2, -G3, -E1, and -E2 can treat problems with fan-in/fan-out ictions. In order to consider the number of levels in a network also as a iction, the transduction procedures based on gate substitution (NETTRA-PG1), gate g (NETTRA-G3), connectable and disconnectable functions (NETTRA-G1, -G2), rror-compensation (NETTRA-E1, -E2) are modified.

This report describes the modifications of these transduction procedures for -restriction and the procedures for designing level-restricted initial networks.

The current version of level-restricted transduction program attempts to design near-optimal multiple-output, multiple-level, loop-free, NOR-gate networks under given fan-in/fan-out restrictions and level restriction.

Words and Document Analysis. 17a. Descriptors

design,
circuits
elements
ems (computers)

ers Open-Ended Terms

ter-aided design	CSPF
rk transduction	level restriction
-restricted transduction	fan-in
optimal networks	fan-out
sible functions	NOR
	NAND

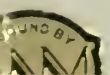
ATI Field/Group

ability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
	20. Security Class (This Page UNCLASSIFIED	22. Price

TIS-35 (10-70)

USCOMM-DC 40329-P71

APR 7 1977
APR 7



UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R no. C002 no.846-851(1977
Level-restricted NOR network transductio
3 0112 088403305